

Year	Number of cases	Number of deaths
1990	100	10
1991	120	12
1992	150	15
1993	180	18
1994	200	20
1995	220	22
1996	250	25
1997	280	28
1998	300	30
1999	320	32
2000	350	35
2001	380	38
2002	400	40
2003	420	42
2004	450	45
2005	480	48
2006	500	50
2007	520	52
2008	550	55
2009	580	58
2010	600	60
2011	620	62
2012	650	65
2013	680	68
2014	700	70
2015	720	72
2016	750	75
2017	780	78
2018	800	80
2019	820	82
2020	850	85

```
RRRRRRRR      SSSSSSSS  TTTTTTTTTT  AAAAAA  CCCCCCCC  CCCCCCCC  EEEEEEEEEEE  SSSSSSSS  SSSSSSSS
RRRRRRRR      SSSSSSSS  TTTTTTTTTT  AAAAAA  CCCCCCCC  CCCCCCCC  EEEEEEEEEEE  SSSSSSSS  SSSSSSSS
RR      RR  SS      TT      AA      AA  CC      CC      EE      SS      SS
RR      RR  SS      TT      AA      AA  CC      CC      EE      SS      SS
RR      RR  SS      TT      AA      AA  CC      CC      EE      SS      SS
RR      RR  SS      TT      AA      AA  CC      CC      EE      SS      SS
RRRRRRRR      SSSSSS      TT      AA      AA  CC      CC      EEEEEEEEE  SSSSSS  SSSSSS
RRRRRRRR      SSSSSS      TT      AA      AA  CC      CC      EEEEEEEEE  SSSSSS  SSSSSS
RR  RR      SS      TT      AAAAAAAAAA  CC      CC      EE      SS      SS
RR  RR      SS      TT      AAAAAAAAAA  CC      CC      EE      SS      SS
RR  RR      SS      TT      AA      AA  CC      CC      EE      SS      SS
RR  RR      SS      TT      AA      AA  CC      CC      EE      SS      SS
RR      RR  SSSSSSSS  TT      AA      AA  CCCCCCCC  CCCCCCCC  EEEEEEEEEEE  SSSSSSSS  SSSSSSSS
RR      RR  SSSSSSSS  TT      AA      AA  CCCCCCCC  CCCCCCCC  EEEEEEEEEEE  SSSSSSSS  SSSSSSSS
                                     ....
                                     ....
                                     ....
                                     ....
```

```
LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```

```

1 0001 0 MODULE RSTACCESS (IDENT = 'V04-000') =
2 0002 0
3 0003 1 BEGIN
4 0004 1
5 0005 1 *****
6 0006 1 *
7 0007 1 *   COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
8 0008 1 *   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
9 0009 1 *   ALL RIGHTS RESERVED.
10 0010 1 *
11 0011 1 *   THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
12 0012 1 *   ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
13 0013 1 *   INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
14 0014 1 *   COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
15 0015 1 *   OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
16 0016 1 *   TRANSFERRED.
17 0017 1 *
18 0018 1 *   THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
19 0019 1 *   AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
20 0020 1 *   CORPORATION.
21 0021 1 *
22 0022 1 *   DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
23 0023 1 *   SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
24 0024 1 *
25 0025 1 *
26 0026 1 *****
27 0027 1
28 0028 1 WRITTEN BY
29 0029 1 Bert Beander June, 1980.
30 0030 1
31 0031 1 MODULE FUNCTION
32 0032 1 This module contains most of the Symbol Table Access routines (except
33 0033 1 for the type routines in module RSTYPES) that the language-specific
34 0034 1 routines call to look up symbols in the Debug Symbol Table and to
35 0035 1 extract symbol table information about those symbols.
36 0036 1
37 0037 1 MODIFIED BY
38 0038 1 Ping Sager
39 0039 1 Rich Title
40 0040 1 Vicki Holt
41 0041 1 Walter Carrell III
42 0042 1
43 0043 1
44 0044 1 REQUIRE 'SRC$:DBGPROLOG.REQ';
45 0178 1
46 0179 1 LIBRARY 'LIB$:DBGGEN.L32';
47 0180 1
48 0181 1 FORWARD ROUTINE
49 0182 1 DBGSADDRESS STRING,      ! Returns ASCII encoding of an address
50 0183 1 DBGSBUILD_INVOC RST,    ! Build Invocation Number RST Entry
51 0184 1 DBGSGET OUTER_REC_ADDRESS, ! Get the outer record's start address from the primary
52 0185 1                               ! pointed to by DBGSGL_CURRENT_PRIMARY
53 0186 1 DBGSGET INNER_REC_ADDRESS, ! Get the inner record's start address from the primary
54 0187 1                               ! pointed to by DBGSGL_CURRENT_PRIMARY
55 0188 1 DBGSIS_IT ENTRY,         ! See if an address is an entry point
56 0189 1 DBGSRST_SHOWSCOPE: NOVALUE, ! Handle the SHOW SCOPE command
57 0190 1 DBGSRST_TEMP_RELEASE: NOVALUE, ! Release all temporary RST entries
```

58	0191	1	
59	0192	1	DBG\$STA_ADDRESS_TO_REGDESCR,
60	0193	1	
61	0194	1	DBG\$STA_GETSOURCEMOD,
62	0195	1	
63	0196	1	DBG\$STA_GETSYMBOL: NOVALUE,
64	0197	1	DBG\$STA_GETSYMOFF,
65	0198	1	DBG\$STA_LINE_NUM RST,
66	0199	1	DBG\$STA_LOCK_SYMID: NOVALUE,
67	0200	1	DBG\$STA_LOOKUP_GBL,
68	0201	1	
69	0202	1	DBG\$STA_NOEVALBIT,
70	0203	1	
71	0204	1	DBG\$STA_NUMBERED_SCOPE: NOVALUE,
72	0205	1	DBG\$STA_RECORD_COMPONENT,
73	0206	1	DBG\$STA_RECORD_INDEX,
74	0207	1	DBG\$STA_REGISTER_NAME,
75	0208	1	DBG\$STA_SAME_DST_OBJECT,
76	0209	1	DBG\$STA_SETCONTEXT: NOVALUE,
77	0210	1	DBG\$STA_SETREGISTERS: NOVALUE,
78	0211	1	DBG\$STA_SYM IS LITERAL,
79	0212	1	DBG\$STA_SYMRIND: NOVALUE,
80	0213	1	DBG\$STA_SYMNAME: NOVALUE,
81	0214	1	DBG\$STA_SYMPARENT,
82	0215	1	DBG\$STA_SYMPATHNAME: NOVALUE,
83	0216	1	DBG\$STA_SYMVALUE: NOVALUE,
84	0217	1	DBG\$STA_UNLOCK_SYMID: NOVALUE,
85	0218	1	DBG\$STA_VALSPEC: NOVALUE,
86	0219	1	DBG\$STA_VARIANT_SELECT,
87	0220	1	DBG\$STA_VARIANT_VALUE,
88	0221	1	
89	0222	1	DBG\$TEST_ROUTINE_CALL,
90	0223	1	DBG\$TRANS TO REGNAME,
91	0224	1	ADD_TO_REF_COUNT: NOVALUE,
92	0225	1	
93	0226	1	CHECK_DUPLICATE,
94	0227	1	EVAL_MAT_SPEC: NOVALUE,
95	0228	1	FOLLOW_STATIC_LINK,
96	0229	1	
97	0230	1	GET_RECORD_ADDRESS,
98	0231	1	GET_REGISTER_SYMID,
99	0232	1	GET_REGISTER_VALUES: NOVALUE,
100	0233	1	
101	0234	1	SCOPE_RULE_COBOL,
102	0235	1	
103	0236	1	SCOPE_RULE_NORMAL,
104	0237	1	
105	0238	1	SCOPE_RULE_PLI,
106	0239	1	
107	0240	1	SETCONTEXT_ERROR_HANDLER,
108	0241	1	STACK_MACHINE: NOVALUE,
109	0242	1	VALSPEC_ERROR_HANDLER,
110	0243	1	VALSPEC_SCOPE_ERROR: NOVALUE,
111	0244	1	VALSPEC_ROUT_CALL: NOVALUE,
112	0245	1	
113	0246	1	VALSPEC_ROUT_CALL_HANDLER;
114	0247	1	

which are not locked
Converts an absolute address to a
Register Descriptor (or zero)
Get Module RST pointer to use for
source line display
Convert pathname to a symbol
Convert address to symbol and offset
Build a Line Number RST Entry
Lock an RST entry in RST memory
Look up a symbol in the image's Global
Symbol Table (the GST)
See if the NOEVAL bit is set in a
symbol's value spec.
Find "numbered" scope from PC in stack
Returns SYMID of N-th record component
Returns index of a record component
Generates print name for a register
See if two SYMIDs refer to same DST
Set up context for value evaluation
Set register values back in save areas
See if symbol is a literal value
Get a symbol's kind
Get a symbol's name
Get parent SYMID for a data component
Get a symbol's full pathname
Get a symbol's value or address
Unlock an RST entry lock in RST memory
Evaluate a DST Value Spec
Return variant entry given tag value
See if tag variable value matches a
specified record variant
Routine to be called for testing stack machine routine calls
Translates address of register
Increment or decrement RST entry ref-
erence count
Check for duplicate RST Entry
Evaluate a Materialization Spec
Follow static links through call stack
for up-level addressing
Get a record start address
Returns SYMID or 0 for register name
Get register values from the current
CALL frame
Select candidate symbol using COBOL
scope rules
Select candidate symbol using "normal"
scope rules
Select candidate symbol using PL/I
scope rules
Error handler for DBG\$STA SETCONTEXT
Value Spec stack machine Interpreter
Error handler for DBG\$STA VALSPEC
Value Spec scope error routine
Do routine call on a compiler-supplied
routine for Value Spec evaluation
A handler to catch the abnormal
status for VALSPEC_ROUT_CALL

116	0248	1	EXTERNAL ROUTINE	
117	0249	1	DBG\$COPY MEMORY	Create a new copy of a memory block
118	0250	1	DBG\$GET_DST_NAME,	Get the ASCII name from a DST record
119	0251	1	DBG\$GET_MEMORY,	Get a permanent memory block
120	0252	1	DBG\$GET_TEMP_MEM,	Get a "temporary" memory block
121	0253	1	DBG\$HASH_FIND,	Find a name in the RST hash table
122	0254	1	DBG\$HASH_FIND SETUP:NOVALUE,	Set up calls on HASH_FIND routine
123	0255	1	DBG\$HASH_INSERT: NOVALUE,	Insert an RST entry in hash table
124	0256	1	DBG\$LINE_TO_PC_LOOKUP,	Look up the PC for a given line number
125	0257	1	DBG\$NOCOPY_DESC,	Copy a primary descriptor
126	0258	1	DBG\$NEWLINE: NOVALUE,	Flush current print line
127	0259	1	DBG\$NGET_RADIX,	Returns present radix
128	0260	1	DBG\$NPATRDESC_TO_CS:NOVALUE,	Generate pathname ASCII string from a
129	0261	1		pathname descriptor
130	0262	1	DBG\$PC_TO_LINE_LOOKUP,	Look up a line number given a PC addr
131	0263	1	DBG\$PRIM_TO_VAL,	Convert a primary to a value
132	0264	1	DBG\$PRINT: NOVALUE,	Print some ASCII text
133	0265	1	DBG\$PRINT CONTROL,	Set up print controls
134	0266	1	DBG\$REL_MEMORY: NOVALUE,	Release a memory block to memory pool
135	0267	1	DBG\$RST_BUILD: NOVALUE,	Build the RST for a specified module
136	0268	1	DBG\$RST_MOST_RECENT:NOVALUE,	Mark a module as being the Most
137	0269	1		Recently Referenced module
138	0270	1	DBG\$SEARCH_GLOBAL,	Tries to symbolize virtual address by
139	0271	1		searching global symbol chain
140	0272	1	DBG\$SEARCH_SAT,	Tries to symbolize virtual address by
141	0273	1		searching SAT.
142	0274	1	DBG\$SEARCH_VAX_CALL_STACK,	Tries to symbolize virtual address by
143	0275	1		searching through call stack.
144	0276	1	DBG\$STA_SYMTYPE : NOVALUE,	Get TYPE of Data Item
145	0277	1	DBG\$STA_TYP_ARRAY : NOVALUE,	Return information about arrays
146	0278	1	DBG\$STA_TYPEFCODE,	Obtain fcode from SYMID
147	0279	1	DBG\$SYMBOLIZE REG,	Finds symbols bound to specified register.
148	0280	1	SYSSFAO: ADDRESSING_MODE (ABSOLUTE);	System service for formatting output
149	0281	1		
150	0282	1	EXTERNAL	
151	0283	1	DBG\$FINAL_HANDL,	Call frame exception handler--used
152	0284	1		searching for a numeric scope
153	0285	1	DBG\$GB_MOD_PTR: REF VECTOR[.BYTE],	Current mode setting
154	0286	1	DBG\$GB_LANGUAGE: BYTE,	The currently SET language code
155	0287	1		
156	0288	1	DBG\$GB_NO_GLOBALS: BYTE,	Number of global symbols in the GST
157	0289	1	DBG\$GB_VERB: BYTE,	Holds command verb
158	0290	1	DBG\$GL_CMND_RADIX,	Radix to use for EXAMINE
159	0291	1	DBG\$GL_CURRENT_PRIMARY,	Pointer to the primary being processed
160	0292	1	DBG\$GV_CONTROL: DBG\$CONTROL_FLAGS,	DEBUG control bits
161	0293	1	DBG\$RUNFRAME: BLOCK[.BYTE],	The current user run frame
162	0294	1	DBG\$PSEUDO_EXIT,	Point to which CALL command CALL re-
163	0295	1		turns--used to find numeric scope
164	0296	1	DST\$BEGIN_ADDR,	Virtual address where the DST begins
165	0297	1	DST\$END_ADDR,	Virtual address of last byte of DST
166	0298	1	LRUM\$MOST_RECENT,	Pointer to the RST entry of the Most
167	0299	1		Recently Referenced module
168	0300	1	RST\$REF_LIST: REF VECTOR[.LONG],	Pointer to list of RST entries refer-
169	0301	1		enced by current Debug command
170	0302	1	RST\$TEMP_LIST,	Pointer to Temporary RST Entry List
171	0303	1	DBG\$REG_VALUES: VECTOR[.LONG],	Vector of user register values in the
172	0304	1		current context

```
173 0305 1   DBG$REG_VECTOR: VECTOR[,LONG],      ! Vector of pointers to user register
174 0306 1   ! save locations in current context
175 0307 1   RST$SET_SCOPE,                      ! Set if called from DBG$RST_SETSCOPE
176 0308 1   RST$START_ADDR: REF RST$ENTRY,      ! Pointer to first Module RST Entry
177 0309 1   SAT$START_ADDR,                    ! Address of first Static Address Table
178 0310 1   ! (SAT) entry on Program SAT chain
179 0311 1   SCOPE$LIST;                        ! Pointer to first Scope List entry
180 0312 1
181 0313 1   OWN
182 0314 1   DBG$REG_SCOPE: INITIAL(0),          ! Numeric scope for context register
183 0315 1   DBG$REG_SYMID: INITIAL(0),         ! SYMID used to set the current context
184 0316 1   DBG$SCOPE_NUMBER: INITIAL(0);      ! Scope number for current context set
185 0317 1   ! by routine DBG$STA_SETCONTEXT
186 0318 1
187 0319 1
188 0320 1   ! Field definitions and declaration macro for the 'candidate block' block-vector
189 0321 1   ! used by DBG$STA_GETSYMBOL and the SCOPE_RULE_xxx routines.
190 0322 1
191 0323 1   FIELD CAND_FLD_DEF =
192 0324 1   SET
193 0325 1   CAND_RSTPTR = [ 0, L_ ],            ! Pointer to symbol's RST entry
194 0326 1   CAND_PINDEX = [ 1, L_ ],          ! Pathname vector index + 1 for symbol
195 0327 1   TES;
196 0328 1
197 0329 1   LITERAL
198 0330 1   CAND_ENTSIZ = 2;                    ! Longword size of a candidate entry
199 0331 1
200 0332 1   MACRO
201 0333 1   CAND_BLOCKVECTOR = BLOCKVECTOR[,CAND_ENTSIZ,LONG] FIELD(CAND_FLD_DEF) %;
202 0334 1
203 0335 1   LITERAL
204 0336 1   Outer = 1,                          ! Flag value for GET_RECORD_ADDRESS to return the outer reco
205 0337 1   Inner = 2;                          ! Flag value for GET_RECORD_ADDRESS to return the inner reco
206 0338 1
207 0339 1   !**
208 0340 1   ! This is a test DST used to test DBG$GET_OUTER_REC_ADDRESS
209 0341 1   ! and DBG$GET_INNER_REC_ADDRESS. To use it, you use the SUPER
210 0342 1   ! DEBUGGER to put the address of the test record in place of
211 0343 1   ! the address of the record you've asked for in DBG$STA_VALSPEC.
212 0344 1   ! Thus, fooling the debugger into using the test record.
213 0345 1   !--
214 0346 1   GLOBAL BIND
215 0347 1   DBG$TEST_DST = UPLIT BYTE (
216 0348 1   DST$K_VS_FOLLOWS,
217 0349 1   WORD(-11),
218 0350 1   DST$K_VS_ALLOC_DYN,
219 0351 1   DST$K_MS_BYTADDR,
220 0352 1   DST$K_MS_MECH_STK,
221 0353 1   0,
222 0354 1   DST$K_STK_PUSHIML,
223 0355 1   LONG(DBG$TEST_ROUTINE_CALL),
224 0356 1   DST$K_STK_RTNCALL,
225 0357 1   DST$K_STK_STOP);
226 0358 1
```

```
228 0359 1 GLOBAL ROUTINE DBG$ADDRESS_STRING (ADDRESS_DESC) =
229 0360 1
230 0361 1 FUNCTION
231 0362 1 This routine accepts an address descriptor and converts the contained
232 0363 1 virtual address (within the address descriptor), ignoring offset, to a
233 0364 1 counted ASCII string, the address of which is returned as the routine
234 0365 1 value. If the address is in the Debugger's register save area, the
235 0366 1 corresponding register name is returned in the counted string. Otherwise,
236 0367 1 the address is returned as a numeric string in the proper radix. If a
237 0368 1 register name is returned, it is preceded by the corresponding scope
238 0369 1 number, for example, '2\%R5' for register R5 in the scope two call frames
239 0370 1 down in the stack. For the top call frame, the scope number is zero. The
240 0371 1 scope number is determined by the last call to DBG$STA_SETCONTEXT.
241 0372 1
242 0373 1 This routine gets the current scope number from the global symbol
243 0374 1 DBG$REG_SCOPE which set up by DBG$STA_SETCONTEXT. It also uses the
244 0375 1 global symbols DBG$GB_VERB which points to the current command being
245 0376 1 processed) and DBG$GL_CMND_RADIX (the radix in effect for an EXAMINE\
246 0377 1 command) to determine the appropriate radix to use.
247 0378 1
248 0379 1 INPUTS
249 0380 1 ADDRESS_DESC - A longword containing the address of an address
250 0381 1 descriptor of a VAX virtual address.
251 0382 1
252 0383 1 OUTPUTS
253 0384 1 The address of a counted ASCII string representing the input address
254 0385 1 is returned as the routine value.
255 0386 1
256 0387 1
257 0388 2 BEGIN
258 0389 2
259 0390 2 MAP
260 0391 2 ADDRESS_DESC: REF DBG$ADDRESS_DESC; ! Pointer to input address descr.
261 0392 2
262 0393 2 LOCAL
263 0394 2 RADIX, ! Radix to use for output
264 0395 2 CONTROL_DESC: BLOCK [8,BYTE], ! $FAO control block
265 0396 2 FAO_LENGTH: WORD, ! $FAO output length
266 0397 2 OUTPUT_DESC: BLOCK [8,BYTE], ! Output descriptor
267 0398 2 OUTPUT_BUFFER: REF VECTOR [,BYTE]; ! Output buffer
268 0399 2
269 0400 2
270 0401 2
271 0402 2 ! Check to see if address can be resymbolized to a register.
272 0403 2
273 0404 2 IF DBG$TRANS_TO_REGNAME (.ADDRESS_DESC [DBG$GL_ADDRESS_BYTE_ADDR],
274 0405 2 OUTPUT_BUFFER)
275 0406 2 THEN
276 0407 2 RETURN .OUTPUT_BUFFER;
277 0408 2
278 0409 2
279 0410 2 ! Register resymbolization not possible. Check to determine what radix to
280 0411 2 use and set up for FAO call.
281 0412 2
282 0413 2 IF .DBG$GB_VERB EQL DBG$K_EXAMINE_VERB
283 0414 2 THEN
284 0415 2 BEGIN
```

```
285 0416 3 RADIX = .DBG$GL CMND RADIX;  
286 0417 3 IF .RADIX EQL DBG$K_DEFAULT  
287 0418 3 THEN  
288 0419 3 RADIX = DBG$NGET_RADIX();  
289 0420 3 END  
290 0421 3 ELSE  
291 0422 3 RADIX = DBG$NGET_RADIX();  
292 0423 3  
293 0424 3 CONTROL_DESC [DSC$A_POINTER] = (  
294 0425 3 CASE .RADIX FROM DBG$K_DEFAULT TO DBG$K_HEX OF  
295 0426 3 SET  
296 0427 3  
297 0428 3 ! Octal radix. Edit the address into ASCII octal.  
298 0429 3  
299 0430 3 [DBG$K_OCTAL]:  
300 0431 3 BEGIN  
301 0432 3 CONTROL_DESC [DSC$W_LENGTH] = %CHARCOUNT ('!OL');  
302 0433 3 UPLIT BYTE ('!OL')  
303 0434 3 END;  
304 0435 3  
305 0436 3  
306 0437 3 ! Hexadecimal radix. Edit the address into hexadecimal.  
307 0438 3  
308 0439 3 [DBG$K_HEX]:  
309 0440 3 BEGIN  
310 0441 3 CONTROL_DESC [DSC$W_LENGTH] = %CHARCOUNT ('!XL');  
311 0442 3 UPLIT BYTE ('!XL')  
312 0443 3 END;  
313 0444 3  
314 0445 3  
315 0446 3 ! Use decimal radix for all other cases. Edit the address into  
316 0447 3 decimal ASCII.  
317 0448 3  
318 0449 3 [INRANGE, OVRANGE]:  
319 0450 3 BEGIN  
320 0451 3 CONTROL_DESC [DSC$W_LENGTH] = %CHARCOUNT ('!UL');  
321 0452 3 UPLIT BYTE ('!UL')  
322 0453 3 END;  
323 0454 3  
324 0455 3  
325 0456 3 TES);  
326 0457 2  
327 0458 2  
328 0459 2 ! Get some storage for the string.  
329 0460 2  
330 0461 2 OUTPUT_BUFFER = DBG$GET_TEMPMEM(5);  
331 0462 2  
332 0463 2  
333 0464 2 ! Call $FAO to do the formatting.  
334 0465 2  
335 0466 2 OUTPUT_DESC [DSC$W_LENGTH] = (5 * %UPVAL) - 2;  
336 0467 2 OUTPUT_DESC [DSC$A_POINTER] = OUTPUT_BUFFER [2];  
337 0468 2 IF NOT $SY$FAO (CONTROL_DESC,  
338 0469 2 FAO_LENGTH,  
339 0470 2 OUTPUT_DESC,  
340 0471 2 .ADDRESS_DESC [DBG$L_ADDRESS_BYTE_ADDR])  
341 0472 2 THEN
```

```

342      0473      2          $DBG_ERROR('RSTACKES\ADDRESS_STRING');
343      0474      2
344      0475      2
345      0476      2      ! The string is formatted, but we want to insert a leading '0' for HEX
346      0477      2      ! when the first character is A, B, C, D, E, or F.
347      0478      2
348      0479      2      IF .RADIY EQL DBG$K_HEX
349      0480      2      THEN
350      0481      3          BEGIN
351      0482      3          IF .OUTPUT_BUFFER [2] GTR '9'
352      0483      3          THEN
353      0484      4              BEGIN
354      0485      4              OUTPUT_BUFFER [1] = '0';
355      0486      4              OUTPUT_BUFFER [0] = .FAO_LENGTH + 1;
356      0487      4              RETURN OUTPUT_BUFFER [0];
357      0488      3          END;
358      0489      3
359      0490      2          END;
360      0491      2
361      0492      2
362      0493      2      ! Just return what $FAO formatted.
363      0494      2
364      0495      2      OUTPUT_BUFFER [1] = .FAO_LENGTH;
365      0496      2      RETURN OUTPUT_BUFFER [1];
366      0497      2
367      0498      1      END;

```

```

.TITLE RSTACCESS
.IDENT \V04-000\

.PSECT DBG$PLIT,NOWRT, SHR, PIC,0

                                FD 00000 P.AAA: .BYTE -3
                                000B 00001      .WORD 11
                        12 00 02 01 02 00003      .BYTE 2, 1, 2, 0, 18
                                00000000v 00008      .ADDRESS DBG$TEST_ROUTINE_CALL
                                17 28 0J00C      .BYTE 40, 23
                                4C 4F 21 0000E P.AAB: .ASCII \!OL\
                                4C 58 21 00011 P.AAC: .ASCII \!XL\
                                4C 55 21 00014 P.AAD: .ASCII \!UL\
52 44 44 41 5C 53 53 45 43 43 41 54 53 52 18 00017 P.AAE: .ASCII <24>\RSTACCESS\<92>\ADDRESS_STRING\
47 4E 49 52 54 53 5F 53 53 45 00026

.PSECT DBG$OWN,NOEXE, PIC,2

J00000000 00000 DBG$REG_SCOPE:
                                .LONG C
000000000 00004 DBG$REG_SYMID:
                                .LONG 0
000000000 00008 DBG$SCOPE_NUMBER:
                                .LONG 0

DBG$TEST_DST== P.AAA
.EXTRN DBG$COPY_MEMORY
.EXTRN DBG$GET_DST_NAME
.EXTRN DBG$GET_MEMORY, DBG$GET_TEMPMEM

```

```
0020      OF      0020      0020      0020      0020
0025      0020      0020      0020      0020      0020
0020      0020      0020      0020      0020      0020
0028      0020      0020      0020      0020      0020

      54 00000000' EF 9E 00002
      5E      18 C2 00009
      04      5E DD 0000C
      0000V CF      BC DD 0000E
      04      02 FB 00011
      50      50 E9 00016
      50      6E D0 00019
      07 00000000G 00 91 0001D 1$:
      53 00000000G 00 D0 00026
      01      53 D1 0002D
      00000000G 00 0A 12 00030
      53      00 FB 00032 2$:
      10 AE      50 D0 00039
      01      03 B0 0003C 3$:
      53 CF 00040
      0020      0020      0020      0020      0020      0020
      0020      0020      0020      0020      0020      0020
      0020      0020      0020      0020      0020      0020
      0020      0020      0020      0020      0020      0020
```

```
.EXTRN DBG$HASH_FIND, DBG$HASH_FIND_SETUP
.EXTRN DBG$HASH_INSERT
.EXTRN DBG$LINE_TO_PC_LOOKUP
.EXTRN DBG$NCOPIY_DESC, DBG$NEWLINE
.EXTRN DBG$NGET_RADIX, DBG$NPATHDESC_TO_CS
.EXTRN DBG$PC_TO_LINE_LOOKUP
.EXTRN DBG$PRIM_TO_VAC
.EXTRN DBG$PRINT, DBG$PRINT_CONTROL
.EXTRN DBG$REL_MEMORY, DBG$RST_BUILD
.EXTRN DBG$RST_MOST_RECENT
.EXTRN DBG$SEARCH_GLOBAL
.EXTRN DBG$SEARCH_SAT, DBG$SEARCH_VAX_CALL_STACK
.EXTRN DBG$STA_SYMTYPE
.EXTRN DBG$STA_TYP_ARRAY
.EXTRN DBG$STA_TYPEFCODE
.EXTRN DBG$SYMBOLIZE_REG
.EXTRN SYSS$FAO, DBG$FINAL_HANDL
.EXTRN DBG$GB_MOD_PTR, DBG$GB_LANGUAGE
.EXTRN DBG$GB_NO_GLOBALS
.EXTRN DBG$GB_VERB, DBG$GL_CMND_RADIX
.EXTRN DBG$GL_CURRENT_PRIMARY
.EXTRN DBG$GV_CONTROL, DBG$RUNFRAME
.EXTRN DBG$PSEUDO_EXIT
.EXTRN DST$BEGIN_ADDR, DST$END_ADDR
.EXTRN LRUM$MOST_RECENT
.EXTRN RST$REF_LIST, RST$TEMP_LIST
.EXTRN DBG$REG_VALUES, DBG$REG_VECTOR
.EXTRN RST$SET_SCOPE, RST$START_ADDR
.EXTRN SAT$START_ADDR, SCOPE$LIST
```

.PSECT DBG\$CODE, NOWRT, SHR, PIC, 0

```
.ENTRY DBG$ADDRESS_STRING, Save R2, R3, R4
MOVAB P.AAD, R4
SUBL2 #24, SP
PUSHL SP
PUSHL @ADDRESS_DESC
CALLS #2, DBG$TRANS_TO_REGNAME
BLBC R0, 1$
MOVL OUTPUT_BUFFER, R0
RET
CMPB DBG$GB_VERB, #7
BNEQ 2$
MOVL DBG$GL_CMND_RADIX, RADIX
CMPL RADIX, #1
BNEQ 3$
CALLS #0, DBG$NGET_RADIX
MOVL R0, RADIX
MOVW #3, CONTROL_DESC
CASEL RADIX, #1, #15
.WORD 5$-4$, -
      5$-4$, -
      5$-4$, -
      5$-4$, -
      5$-4$, -
      5$-4$, -
      5$-4$, -
      5$-4$, -
```

```
: 0359
:
: 0404
:
: 0407
:
: 0413
:
: 0416
: 0417
:
: 0422
:
: 0452
: 0425
:
```

						6\$-4\$,-				
						5\$-4\$,-				
						5\$-4\$,-				
						5\$-4\$,-				
						5\$-4\$,-				
						5\$-4\$,-				
						5\$-4\$,-				
						5\$-4\$,-				
						7\$-4\$,-				
	50		64	9E	00064	5\$:	MOVAB	P.AAD, R0	0453	
			0A	11	00067		BRB	8\$		
	50	FA	A4	9E	00069	6\$:	MOVAB	P.AAB, R0	0434	
			04	11	0006D		BRB	8\$		
	50	FD	A4	9E	0006F	7\$:	MOVAB	P.AAC, R0	0443	
14	AE		50	D0	00073	8\$:	MOVL	R0, CONTROL_DESC+4	0424	
			05	DD	00077		PUSHL	#5	0461	
00000000G	00		01	FB	00079		CALLS	#1, DBG\$GET TEMPMEM		
	6E		50	D0	00080		MOVL	R0, OUTPUT_BUFFER		
08	AE		12	B0	00083		MOVW	#18, OUTPUT_DESC	0466	
	52		6E	D0	00087		MOVL	OUTPUT_BUFFER, R2	0467	
0C	AE	02	A2	9E	0008A		MOVAB	2(R2), OUTPUT_DESC+4		
		04	BC	DD	0008F		PUSHL	@ADDRESS_DESC	0471	
		0C	AE	9F	00092		PUSHAB	OUTPUT_DESC	0468	
		0C	AE	9F	00095		PUSHAB	FAO_LENGTH		
		1C	AE	9F	00098		PUSHAB	CONTROL_DESC		
00000000G	9F		04	FB	0009B		CALLS	#4, @SYSS\$FAO		
	12		50	E8	000A2		BLBS	R0, 9\$		
		03	A4	9F	000A5		PUSHAB	P.AAE	0473	
			01	DD	000A8		PUSHL	#1		
		00028362	8F	DD	000AA		PUSHL	#164706		
00000000G	00		03	FB	000B0		CALLS	#3, LIB\$SIGNAL		
	10		53	D1	000B7	9\$:	CMPL	RADIX, #16	0479	
			13	12	000BA		BNEQ	10\$		
	39	02	A2	91	000BC		CMPL	2(R2), #57	0482	
			0D	1B	000C0		BLEQU	10\$		
	01	A2	30	90	000C2		MOVW	#48, 1(R2)	0485	
62	04	AE	01	81	000C6		ADDB3	#1, FAO_LENGTH, (R2)	0486	
	50		52	D0	000CB		MOVL	R2, R0	0487	
				04	000CE		RET			
	01	A2	04	AE	90	000CF	10\$:	MOVW	FAO_LENGTH, 1(R2)	0495
	50	01	A2	9E	000D4		MOVAB	1(R2), R0	0496	
			04	000D8			RET		0498	

; Routine Size: 217 bytes, Routine Base: DBG\$CODE + 0000

```

369 0499 1 GLOBAL ROUTINE DBGSBUILD_INVOC_RST(OLDRST, INVOCNUM) =
370 0500 1
371 0501 1 FUNCTION
372 0502 1 This routine builds an RST entry with an attached invocation number for
373 0503 1 a specified symbol. To do this, it accepts the symbol's RST entry as
374 0504 1 input and creates a new copy of that RST entry. The RST$V_INVOCNUM flag
375 0505 1 is set in the new copy. It then builds an Invocation Number RST Entry
376 0506 1 to hold the actual invocation number. The RST$L_SYMCHNPTR field in the
377 0507 1 new symbol entry is set to point to the Invocation Number RST Entry.
378 0508 1 Both new RST entries are added to the Temporary RST Entry List pointed
379 0509 1 to by RST$TEMP_LIST. The new symbol RST entry represents this specific
380 0510 1 invocation of the new symbol, and its address is returned to the caller.
381 0511 1
382 0512 1 INPUTS
383 0513 1 OLDRST - Pointer to the RST entry of the symbol to which an invocation
384 0514 1 number should be attached.
385 0515 1
386 0516 1 INVOCNUM - The desired invocation number. This is assumed to be applied
387 0517 1 to the inner-most invocable entity (routine) in the scope in
388 0518 1 which the OLDRST symbol is declared.
389 0519 1
390 0520 1 OUTPUTS
391 0521 1 A pointer to the new symbol RST entry (which includes the invocation
392 0522 1 information) is returned as the routine value.
393 0523 1
394 0524 1
395 0525 2 BEGIN
396 0526 2
397 0527 2 MAP
398 0528 2 OLDRST: REF RST$ENTRY; ! Pointer to the symbol RST entry
399 0529 2
400 0530 2 OWN
401 0531 2 RST_SIZE_TBL: ! RST entry size look-up table indexed
402 0532 2 VECTOR[RST$K_KIND_MAXIMUM + 1, BYTE] ! by entry kind
403 0533 2 PRESET( [RST$K_INVALID] = 0,
404 0534 2 [RST$K_NOTUNIQUE] = 0,
405 0535 2 [RST$K_MODULE] = 0,
406 0536 2 [RST$K_ROUTINE] = RST$K_ROUTENTSIZ,
407 0537 2 [RST$K_BLOCK] = RST$K_LEXENTSIZ,
408 0538 2 [RST$K_ENTRY] = RST$K_EPTENTSIZ,
409 0539 2 [RST$K_LABEL] = RST$K_LBLENTSIZ,
410 0540 2 [RST$K_LINE] = RST$K_LINENTSIZ,
411 0541 2 [RST$K_DATA] = RST$K_DATENTSIZ,
412 0542 2 [RST$K_TYPCOMP] = RST$K_DATENTSIZ,
413 0543 2 [RST$K_TYPE] = 0,
414 0544 2 [RST$K_VARIANT] = 0,
415 0545 2 [RST$K_INVOCNUM] = 0,
416 0546 2 [RST$K_OVERLOAD] = 0);
417 0547 2
418 0548 2 LOCAL
419 0549 2 INVPTR: REF RST$ENTRY; ! Pointer to Invocation Number RST Entry
420 0550 2 NEWNST: REF RST$ENTRY; ! Pointer to new copy of symbol RST
421 0551 2 SIZE; ! The size of this RST entry
422 0552 2
423 0553 2
424 0554 2
425 0555 2 ! Determine the size and validity of the symbol RST entry. We do not accept
```



```

: 426      0556      2      ! Module, Type, or Variant RST entries.
: 427      0557      2
: 428      0558      2      SIZE = 0;
: 429      0559      2      IF (.OLDRST[RST$B_KIND] GEQ RST$K_KIND_MINIMUM) AND
: 430      0560      2      (.OLDRST[RST$B_KIND] LEQ RST$K_KIND_MAXIMUM)
: 431      0561      2      THEN
: 432      0562      2          SIZE = .RST_SIZE_TBL[.OLDRST[RST$B_KIND]];
: 433      0563      2
: 434      0564      2      IF .SIZE EQL 0 THEN $DBG_ERROR('RSTACK\DBG$BUILD_INVOC_RST');
: 435      0565      2
: 436      0566      2
: 437      0567      2      ! Copy the symbol's RST entry (the "old" RST entry) into a new memory
: 438      0568      2      ! block (the "new" RST entry). Note that we copy the whole memory block
: 439      0569      2      ! so that any embedded DST entry is copied also. Then fill in all fields
: 440      0570      2      ! in the new entry that must be different from those in the old entry.
: 441      0571      2
: 442      0572      2      NEWRST = DBG$COPY_MEMORY(.OLDRST);
: 443      0573      2      NEWRST[RST$L_HASH_BLINK] = 0;
: 444      0574      3      IF .OLDRST[RST$L_DSTPTR] EQL (.OLDRST + 4*.SIZE)
: 445      0575      2      THEN
: 446      0576      2          NEWRST[RST$L_DSTPTR] = .NEWRST + 4*.SIZE;
: 447      0577      2
: 448      0578      2      NEWRST[RST$W_REFCOUNT] = 0;
: 449      0579      2
: 450      0580      2
: 451      0581      2      ! Put the new symbol entry on the Temporary RST Entry List.
: 452      0582      2
: 453      0583      2      NEWRST[RST$L_HASH_FLINK] = .RST$TEMP_LIST;
: 454      0584      2      RST$TEMP_LIST = .NEWRST;
: 455      0585      2
: 456      0586      2
: 457      0587      2      ! Now build the Invocation Number RST Entry to go with the new symbol entry.
: 458      0588      2      ! Also put it on the Temporary RST Entry List.
: 459      0589      2
: 460      0590      2      INVPTR = DBG$GET_MEMORY(RST$K_INVENTSIZ);
: 461      0591      2      INVPTR[RST$L_UPSCOPEPTR] = .OLDRST;
: 462      0592      2      INVPTR[RST$B_KIND] = RST$K_INVOCNUM;
: 463      0593      2      INVPTR[RST$L_INVOCNUM] = .INVOCNUM;
: 464      0594      2      INVPTR[RST$L_HASH_FLINK] = .RST$TEMP_LIST;
: 465      0595      2      RST$TEMP_LIST = .INVPTR;
: 466      0596      2
: 467      0597      2
: 468      0598      2      ! Finally put a link in the symbol's new RST entry which points to the
: 469      0599      2      ! Invocation Number RST Entry. Then return the address of the new entry.
: 470      0600      2
: 471      0601      2      NEWRST[RST$L_SYMCHNPTR] = .INVPTR;
: 472      0602      2      NEWRST[RST$V_INVOCNUM] = TRUE;
: 473      0603      2      RETURN .NEWRST;
: 474      0604      2
: 475      0605      1      END;
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

```
24 47 42 44 5C 53 53 45 43 43 41 54 53 52 1D 00030 P.AAF: .ASCII <29>\RSTACK\<92>\DBG$BUILD_INVOC_RS\
53 52 5F 43 4F 56 4E 49 5F 44 4C 49 55 42 0003F
```

54 0004D .ASCII \T\ ;
.PSECT DBG\$OWN,NOEXE, PIC,2

00 00 00 07 00 0B 00 07 0B 07 0B 0B 00 00 0000C RST_SIZE_TBL:
_BYTE 0, 0, 11, 8, 7, 8, 7, 0, 11, 0, 7, 0, 0, 0 ;

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0
.ENTRY DBG\$BUILD_INVOC_RST, Save R2,R3,R4,R5 ; 0499
MOVAB RST\$TEMP_LIST, R5 ; 0558
CLRL SIZE ; 0560
MOVL OLDRST, R4
MOVZBL 20(R4), R0
CMPB R0, #13
BGTRU 1\$
MOVZBL RST_SIZE_TBL[R0], SIZE ; 0562
TSTL SIZE ; 0564
BNEQ 2\$
PUSHAB P.AAF
PUSHL #1
PUSHL #164706
CALLS #3, LIB\$SIGNAL ; 0572
PUSHL R4
CALLS #1, DBG\$COPY_MEMORY
MOVL R0, NEWST
CLRL 4(NEWST) ; 0573
MOVAL (R4)[SIZE], R0 ; 0574
CMPL 12(R4), R0
BNEQ 3\$
MOVAL (NEWST)[SIZE], 12(NEWST) ; 0576
CLRW 22(NEWST) ; 0578
MOVL RST\$TEMP_LIST, (NEWST) ; 0583
MOVL NEWST, RST\$TEMP_LIST ; 0584
PUSHL #7 ; 0590
CALLS #1, DBG\$GET_MEMORY
MOVL R4, 16(INVPTR) ; 0591
MOVB #12, 20(INVPTR) ; 0592
MOVL INVOCNUM, 24(INVPTR) ; 0593
MOVL RST\$TEMP_LIST, (INVPTR) ; 0594
MOVL INVPTR, RST\$TEMP_LIST ; 0595
MOVL INVPTR, 8(NEWST) ; 0601
BISB2 #4, 21(NEWST) ; 0602
MOVL NEWST, R0 ; 0603
RET ; 0605

; Routine Size: 136 bytes, Routine Base: DBG\$CODE + 00D9

```

: 477 0606 1 GLOBAL ROUTINE DBG$GET_INNER_REC_ADDRESS( Primptr ) =
: 478 0607 1
: 479 0608 1 FUNCTION
: 480 0609 1     DBG$GET_INNER_REC_ADDRESS returns the address of the inner record
: 481 0610 1     based on the primary pointer passed.
: 482 0611 1     It's called from the stack machine when the value of a record's field
: 483 0612 1     depends on the contents of the record.
: 484 0613 1
: 485 0614 1 INPUTS
: 486 0615 1     Primptr - A pointer to a primary descriptor passed by value.
: 487 0616 1
: 488 0617 1 OUTPUTS
: 489 0618 1     The address of the inner record.
: 490 0619 1
: 491 0620 1 SIDE EFFECTS
: 492 0621 1     Errors may be signaled
: 493 0622 1
: 494 0623 2 BEGIN
: 495 0624 2
: 496 0625 2     RETURN GET_RECORD_ADDRESS( .Primptr, Inner );
: 497 0626 2
: 498 0627 1     END;
```

		0000 00000	.ENTRY	DBG\$GET_INNER_REC_ADDRESS, Save nothing	: 0606
		02 DD 00002	PUSHL	#2	: 0625
		04 AC DD 00004	PUSHL	PRIMPTR	:
0000V CF		02 FB 00007	CALLS	#2, GET_RECORD_ADDRESS	:
		04 0000C	RET		: 0627

; Routine Size: 13 bytes, Routine Base: DBG\$CODE + 0161

```
: 500      0628 1 GLOBAL ROUTINE DBG$GET_OUTER_REC_ADDRESS( Primptr ) =
: 501      0629 1
: 502      0630 1 FUNCTION
: 503      0631 1     DBG$GET_OUTER_REC_ADDRESS returns the address of the outer record
: 504      0632 1     based on the primary pointer passed.
: 505      0633 1     It's called from the stack machine when the value of a record's field
: 506      0634 1     depends on the contents of the record.
: 507      0635 1
: 508      0636 1 INPUTS
: 509      0637 1     Primptr - A pointer to a primary descriptor passed by value.
: 510      0638 1
: 511      0639 1 OUTPUTS
: 512      0640 1     The address of the outer record.
: 513      0641 1
: 514      0642 1 SIDE EFFECTS
: 515      0643 1     Errors may be signaled
: 516      0644 1
: 517      0645 2 BEGIN
: 518      0646 2
: 519      0647 2     RETURN GET_RECORD_ADDRESS( .Primptr, Outer );
: 520      0648 2
: 521      0649 1 END;
```

		0000 00000	.ENTRY	DBG\$GET_OUTER_REC_ADDRESS, Save nothing	: 0628
		01 DD 00002	PUSHL	#1	: 0647
		04 AC DD 00004	PUSHL	PRIMPTR	:
0000V Cf		02 FB 00007	CALLS	#2, GET_RECORD_ADDRESS	:
		04 0000C	RET		: 0649

; Routine Size: 13 bytes, Routine Base: DBG\$CODE + 016E

```
0650 1 GLOBAL ROUTINE DBG$IS_IT_ENTRY(ADDR) =
0651 1
0652 1 FUNCTION
0653 1     This routine decides whether a given virtual address in the user program
0654 1     is the address of a CALL entry mask. It returns TRUE if the given ad-
0655 1     dress is the start address of a routine or entry point callable with the
0656 1     CALLS and CALLG instructions. It returns FALSE in all other cases.
0657 1
0658 1     This routine is called in the processing of breakpoints because if a
0659 1     breakpoint is set on a CALL routine (as opposed to a JSB routine or an
0660 1     ordinary code location), the BPT instruction must be placed two bytes
0661 1     past the routine address so it falls on the first instruction instead
0662 1     of the entry mask.
0663 1
0664 1 INPUTS
0665 1     ADDR      - The input address. This routine will determine whether this
0666 1                 address points to an entry mask or not.
0667 1
0668 1 OUTPUTS
0669 1     The routine returns TRUE if ADDR is the address of an entry mask, i.e.
0670 1     is the address of a CALLS or CALLG routine or entry point.
0671 1     The routine returns FALSE otherwise.
0672 1
0673 1 BEGIN
0674 2
0675 2 LOCAL
0676 2     DSTPTR: REF DST$RECORD,           ! Pointer to Routine Begin DST Record
0677 2     GSTPTR: REF RST$ENTRY,           ! Pointer to Global Symbol Table record
0678 2     MODRSTPTR: REF RST$ENTRY,        ! Pointer to current Module RST Entry
0679 2     PROG_SATPTR: REF SAT$ENTRY,      ! Pointer to Program SAT entry
0680 2     RSTPTR: REF RST$ENTRY,           ! Pointer to current RST entry
0681 2     SATPTR: REF SAT$ENTRY;           ! Pointer to SAT entry for a symbol
0682 2
0683 2
0684 2 ! Search through the Program Static Address Table. Here we are looking for
0685 2 ! a module which covers the ADDR address.
0686 2
0687 2 !
0688 2
0689 2 PROG_SATPTR = .SAT$START_ADDR;
0690 2 WHILE .PROG_SATPTR NEQ 0 DO
0691 2 BEGIN
0692 3
0693 3 ! If the current Static Address Table entry is past the address we are
0694 3 ! looking for (has a higher start address), we exit the search loop
0695 3 ! without finding a suitable SAT entry. This works because the Static
0696 3 ! Address Table is sorted on the start address.
0697 3
0698 3 IF .PROG_SATPTR[SAT$START] GTRA .ADDR THEN EXITLOOP;
0699 3
0700 3
0701 3 ! If ADDR is in the range of this SAT entry, we go to the corresponding
0702 3 ! Module RST Entry and search that module's Static Address Table.
0703 3
0704 3 IF .PROG_SATPTR[SAT$END] GEQA .ADDR
0705 3 THEN
0706 3
```

```
580 0707 4
581 0708 4
582 0709 4
583 0710 4
584 0711 4
585 0712 4
586 0713 4
587 0714 4
588 0715 4
589 0716 4
590 0717 5
591 0718 5
592 0719 5
593 0720 5
594 0721 5
595 0722 5
596 0723 5
597 0724 5
598 0725 5
599 0726 5
600 0727 5
601 0728 5
602 0729 5
603 0730 5
604 0731 5
605 0732 5
606 0733 6
607 0734 6
608 0735 6
609 0736 6
610 0737 6
611 0738 6
612 0739 6
613 0740 7
614 0741 7
615 0742 7
616 0743 8
617 0744 7
618 0745 7
619 0746 7
620 0747 6
621 0748 6
622 0749 6
623 0750 6
624 0751 6
625 0752 6
626 0753 6
627 0754 6
628 0755 6
629 0756 6
630 0757 6
631 0758 6
632 0759 6
633 0760 7
634 0761 7
635 0762 7
636 0763 7
```

BEGIN

```
: Loop through this module's SAT looking for a symbol whose address
: matches the desired ADDR address. If we find such a symbol, we
: see if it is an entry point.
```

```
MODRSTPTR = .PROG SATPTR[SAT$L_RSTPTR];
SATPTR = .MODRSTPTR[RST$L_SAT_PTR];
WHILE .SATPTR NEQ 0 DO
  BEGIN
```

```
: If this SAT entry is past the desired address, exit this
: loop--there is no symbol for the address in this module.
: (Again, this Static Address Table is sorted on start address.)
```

```
IF .SATPTR[SAT$L_START] GTRA .ADDR THEN EXITLOOP;
```

```
: If this SAT entry has exactly the start address we want, we
: return TRUE if the corresponding symbol is a CALL routine,
: an entry point, or 'entry mask' data type.
```

```
IF .SATPTR[SAT$L_START] EQLA .ADDR
THEN
```

```
  BEGIN
    RSTPTR = .SATPTR[SAT$L_RSTPTR];
```

```
    : Check for a CALL routine.
```

```
    IF .RSTPTR[RST$B_KIND] EQL RST$K_ROUTINE
    THEN
```

```
      BEGIN
        DSTPTR = .RSTPTR[RST$L_DSTPTR];
        IF (.DSTPTR[DST$B_TYPE] EQL DST$K_RTNBEG) AND
            (NOT .DSTPTR[DST$V_RTNBEG_NO_CALL])
        THEN
          RETURN TRUE;
```

```
      END;
```

```
    : Check for an entry point.
```

```
    IF .RSTPTR[RST$B_KIND] EQL RST$K_ENTRY THEN RETURN TRUE;
```

```
    : Check for data of type ZEM (entry mask). This arises
    : if routines are passed as parameters to other routines.
    : This situation also arises when routine names are
    : imported in PASCAL from environment files.
```

```
    IF .RSTPTR[RST$B_KIND] EQL RST$K_DATA
    THEN
```

```
      BEGIN
        DSTPTR = .RSTPTR[RST$L_DSTPTR];
        IF .DSTPTR[DST$B_TYPE] EQL DST$K_DTYPE_ZEM
        THEN
```

```

637 0764 7 RETURN TRUE;
638 0765 6 END;
639 0766 5 END;
640 0767 5
641 0768 5
642 0769 5 ! Loop for the next symbol SAT entry in this module.
643 0770 5 !
644 0771 5 SATPTR = .SATPTR[SAT$L_FLINK];
645 0772 4 END;
646 0773 4
647 0774 3 END; ! End of If for ADDR in SAT entry range
648 0775 3
649 0776 3
650 0777 3 ! Address not found in this module--loop for the next Program Static
651 0778 3 Address Table entry.
652 0779 3
653 0780 3 PROG_SATPTR = .PROG_SATPTR[SAT$L_FLINK];
654 0781 3 END;
655 0782 3
656 0783 3
657 0784 2 ! A CALL routine or entry point symbol for the address could not be found
658 0785 2 anywhere in the Static Address Table or in any SET module. We therefore
659 0786 2 have to search the Global Symbol Table to see if a symbol for the address
660 0787 2 is declared there.
661 0788 2
662 0789 2 GSTPTR = .RST$START_ADDR;
663 0790 2 WHILE TRUE DO
664 0791 2 BEGIN
665 0792 2
666 0793 2 ! Get a pointer to the next GST entry. Exit loop if there are no more.
667 0794 2 !
668 0795 2 IF .GSTPTR EQL 0 THEN EXITLOOP;
669 0796 2
670 0797 2
671 0798 2 ! If this is a Routine or Entry GST Record for the exact address we
672 0799 2 are looking for, return TRUE -- the address is an entry point.
673 0800 2 !
674 0801 2 IF .GSTPTR[RST$B_KIND] EQL RST$K_ROUTINE
675 0802 2 THEN
676 0803 2 BEGIN
677 0804 2 IF .GSTPTR[RST$L_STARTADDR] EQLA .ADDR THEN RETURN TRUE;
678 0805 2 END;
679 0806 2
680 0807 2
681 0808 2 ! Get next GST entry by following the RST symbol chain pointer.
682 0809 2 !
683 0810 2 GSTPTR = .GSTPTR[RST$L_SYMCHNPTR];
684 0811 2 END;
685 0812 2
686 0813 2
687 0814 2 ! We did not find an entry or procedure definition for the address in the
688 0815 2 GST either. It is thus not an entry point and we return FALSE.
689 0816 2 !
690 0817 2 RETURN FALSE;
691 0818 2
692 0819 2 END;
693 0820 1
```

			003C 00000	.ENTRY	DBG\$IS_IT_ENTRY, Save R2,R3,R4,R5	: 0650
52	00000000G	00	D0 00002	MOVL	SAT\$START_ADDR, PROG_SATPTR	: 0689
54	04	AC	D0 00009	MOVL	ADDR, R4	: 0699
		52	D5 0000D	1\$: TSTL	PROG_SATPTR	: 0690
		58	13 0000F	BEQL	6\$	
54	04	A2	D1 00011	CMPL	4(PROG_SATPTR), R4	: 0699
		52	1A 00015	BGTRU	6\$	
54	08	A2	D1 00017	CMPL	8(PROG_SATPTR), R4	: 0705
		47	1F 0001B	BLSSU	5\$	
55	0C	A2	D0 0001D	MOVL	12(PROG_SATPTR), MODRSTPTR	: 0714
51	18	A5	D0 00021	MOVL	24(MODRSTPTR), SATPTR	: 0715
		3D	13 00025	2\$: BEQL	5\$: 0716
54	04	A1	D1 00027	CMPL	4(SATPTR), R4	: 0724
		37	1A 0002B	BGTRU	5\$	
		30	12 0002D	BNEQ	4\$: 0731
50	0C	A1	D0 0002F	MOVL	12(SATPTR), RSTPTR	: 0734
02	14	A0	91 00033	CM B	20(RSTPTR), #2	: 0738
		10	12 00037	B' 0	3\$	
53	0C	A0	D0 00039	M VL	12(RSTPTR), DSTPTR	: 0741
BE 8F	01	A3	91 0003D	CM PB	1(DSTPTR), #190	: 0742
		05	12 00042	BNEQ	3\$	
	02	A3	95 00044	TSTB	2(DSTPTR)	: 0743
		35	18 00047	BGEQ	8\$	
08	14	A0	91 00049	3\$: CMPB	20(RSTPTR), #8	: 0751
		2F	13 0004D	BEQL	8\$	
06	14	A0	91 0004F	CMPB	20(RSTPTR), #6	: 0758
		0A	12 00053	BNEQ	4\$	
53	0C	A0	D0 00055	MOVL	12(RSTPTR), DSTPTR	: 0761
17	01	A3	91 00059	CMPB	1(DSTPTR), #23	: 0762
		1F	13 0005D	BEQL	8\$	
51		61	D0 0005F	4\$: MOVL	(SATPTR), SATPTR	: 0771
		C1	11 00062	BRB	2\$: 0716
52		62	D0 00064	5\$: MOVL	(PROG_SATPTR), PROG_SATPTR	: 0780
		A4	11 00067	BRB	1\$: 0690
51	00000000G	00	D0 00069	6\$: MOVL	RST\$START_ADDR, GSTPTR	: 0789
		16	13 00070	7\$: BEQL	10\$: 0796
02	14	A1	91 00072	CMPB	20(GSTPTR), #2	: 0802
		0A	12 00076	BNEQ	9\$	
54	18	A1	D1 00078	CMPL	24(GSTPTR), R4	: 0805
		04	12 0007C	BNEQ	9\$	
50		01	D0 0007E	8\$: MOVL	#1, R0	
		04	00081	RET		
51	08	A1	D0 00082	9\$: MOVL	8(GSTPTR), GSTPTR	: 0811
		E8	11 00086	BRB	7\$: 0790
		50	D4 00088	10\$: CLRL	R0	: 0818
		04	0008A	RET		: 0820

; Routine Size: 139 bytes. Routine Base: DBG\$CODE + 017B


```

695 0821 1 GLOBAL ROUTINE DBG$RST_SHOWSCOPE: NOVALUE =
696 0822 1
697 0823 1 FUNCTION
698 0824 1     This routine does most of the work of handling the SHOW SCOPE command.
699 0825 1     It goes through the internal Scope List, and for each scope entry it
700 0826 1     prints out the corresponding scope name.
701 0827 1
702 0828 1 INPUTS
703 0829 1     NONE
704 0830 1
705 0831 1 OUTPUTS
706 0832 1     The SHOW SCOPE response (i.e., 'scope: scope-list') is printed out.
707 0833 1     No value is returned.
708 0834 1
709 0835 1
710 0836 2 BEGIN
711 0837 2
712 0838 2 LOCAL
713 0839 2     INVOCNUM,           ! Invocation number for numeric scope
714 0840 2     MODRSTPTR,       ! Return parameter not actually used
715 0841 2     PATHNAME,      ! Pointer to Pathname Descriptor
716 0842 2     PATH_STRING,   ! Pointer to pathname counted string
717 0843 2     RSTPTR,         ! Pointer to scope RST entry
718 0844 2     SCOPE: REF SCOPE$ENTRY; ! Pointer to current scope list entry
719 0845 2
720 0846 2
721 0847 2
722 0848 2 ! Print the initial "scope: " text.
723 0849 2
724 0850 2 DBG$PRINT(UPLIT BYTE(%ASCIC 'scope: '), 0);
725 0851 2
726 0852 2
727 0853 2 ! Loop through all the Scope List entries until we find the all-set-modules
728 0854 2 ! scope.  For each scope, print out the scope name.
729 0855 2
730 0856 2 SCOPE = .SCOPE$LIST;
731 0857 2 WHILE .SCOPE[SCOPE$L_STATE] NEQ SCOPE$K_SETMODS DO
732 0858 2 BEGIN
733 0859 2
734 0860 2
735 0861 2 ! Print the comma between scope entries.
736 0862 2
737 0863 2 IF .SCOPE NEQ .SCOPE$LIST
738 0864 2 THEN
739 0865 2     DBG$PRINT(UPLIT BYTE(%ASCIC ', '), 0);
740 0866 2
741 0867 2
742 0868 2 ! Now do a CASE on the kind of Scope List entry this is.
743 0869 2
744 0870 2 CASE .SCOPE[SCOPE$L_STATE] FROM SCOPE$K_NORMAL TO SCOPE$K_SETMODS OF
745 0871 2 SET
746 0872 2
747 0873 2
748 0874 2 ! Handle normal scopes as set with the SET SCOPE command. Convert
749 0875 2 ! the scope to a Pathname Descriptor; then convert that to a counted
750 0876 2 ! ASCII string and print that string.
751 0877 2
```

```

: 752      0878 3
: 753      0879 4
: 754      0880 4
: 755      0881 4
: 756      0882 4
: 757      0883 3
: 758      0884 3
: 759      0885 3
: 760      0886 3
: 761      0887 3
: 762      0888 3
: 763      0889 3
: 764      0890 3
: 765      0891 4
: 766      0892 4
: 767      0893 4
: 768      0894 4
: 769      0895 4
: 770      0896 4
: 771      0897 5
: 772      0898 5
: 773      0899 5
: 774      0900 5
: 775      0901 5
: 776      0902 5
: 777      0903 5
: 778      0904 5
: 779      0905 4
: 780      0906 4
: 781      0907 3
: 782      0908 3
: 783      0909 3
: 784      0910 3
: 785      0911 3
: 786      0912 3
: 787      0913 3
: 788      0914 3
: 789      0915 3
: 790      0916 3
: 791      0917 3
: 792      0918 3
: 793      0919 3
: 794      0920 3
: 795      0921 3
: 796      0922 3
: 797      0923 3
: 798      0924 3
: 799      0925 3
: 800      0926 3
: 801      0927 3
: 802      0928 2
: 803      0929 2
: 804      0930 2
: 805      0931 2
: 806      0932 2
: 807      0933 2
: 808      0934 2

[SCOPE$K NORMAL]:
BEGIN
DBG$STA SYMPATHNAME(.SCOPE[SCOPE$L RSTPTR], PATHNAME);
DBG$NPATHDESC TO CS(.PATHNAME, PATH_STRING);
DBG$PRINT(UPLIT BYTE(%ASCIC '!AC'), .PATH_STRING);
END;

: Handle 'numbered' scopes, i.e. scopes relative to the current top
: of the call stack. Here we first print the number itself and then
: the actual scope this corresponds to at present.
[SCOPE$K NUMBERED]:
BEGIN
DBG$PRINT(UPLIT BYTE(%ASCIC '!SL'), .SCOPE[SCOPE$L_MODPTR]);
DBG$STA_NUMBERED_SCOPE(.SCOPE[SCOPE$L_MODPTR],
MODRSTPTR, RSTPTR, INVOCNUM);
IF .RSTPTR NEQ 0
THEN
BEGIN
IF .INVOCNUM NEQ 0
THEN
RSTPTR = DBG$BUILD_INVOC_RST(.RSTPTR, .INVOCNUM);

DBG$STA SYMPATHNAME(.RSTPTR, PATHNAME);
DBG$NPATHDESC TO CS(.PATHNAME, PATH_STRING);
DBG$PRINT(UPLIT BYTE(%ASCIC ' [ = !AC ]'), .PATH_STRING);
END;
END;

: Handle the Global scope. This is done by simply printing '\'.
[SCOPE$K GLOBAL]:
DBG$PRINT(UPLIT BYTE(%ASCIC '\'), 0);

: Handle the all SET modules scope. Since we should never get here,
: we just signal an error.
[SCOPE$K SETMODS]:
$DBG_ERKOR('RSTACCESS\SHOWSCOPE');

TES;

: Link on to the next Scope List entry and loop for the next scope.
SCOPE = .SCOPE[SCOPE$L_FLINK];
END;

: We are all done. Flush the output buffer and return.
DBG$NEWLINE();
RETURN;
```

: 809
: 810 0935 2
0936 1 END;

```
.PSECT DBG$PLIT,NOWRT, SHR, PIC,0

20 3A 65 70 6F 63 73 07 0004E P.AAG: .ASCII <7>\scope: \
43 20 2C 02 00056 P.AAH: .ASCII <2>\, \
41 21 03 00059 P.AAI: .ASCII <3>\!AC\
4C 53 21 03 0005D P.AAJ: .ASCII <3>\!SL\
5D 20 43 41 21 20 3D 20 5B 20 0A 00061 P.AAK: .ASCII <10>\ [ = !AC ]\
5C 01 0006C P.AAL: .ASCII <1><92>
57 4F 48 53 5C 53 53 45 43 43 41 54 53 52 13 0006E P.AAM: .ASCII <19>\RSTACCESS\<92>\SHOWSCOPE\
45 50 4F 43 53 0007D

.PSECT DBG$CODE,NOWRT, SHR, PIC,0

007C 00000 .ENTRY DBG$RST SHOWSCOPE, Save R2,R3,R4,R5,R6 : 0821
56 00000000G 00 9E 00002 MOVAB DBG$NPATHDESC_TO_CS, R6
55 00000000G 00 9E 00009 MOVAB SCOPE$LIST, R5
54 00000000G 00 9E 00010 MOVAB DBG$PRINT, R4
53 00000000G EF 9E 00017 MOVAB P.AAG, R3
5E 14 C2 0001E SUBL2 #20, SP
7E D4 00021 CLRL -(SP) : 0850
53 DD 00023 PUSHL R3
64 02 FB 00025 CALLS #2, DBG$PRINT
52 65 D0 00028 MOVL SCOPE$LIST, SCOPE : 0856
04 04 A2 D1 0002B 1$: CMPL 4(SCOPE), #4 : 0857
03 12 0002F BNEQ 2$
00A4 31 00031 BRW 12$
65 52 D1 00034 2$: CMPL SCOPE, SCOPE$LIST : 0863
08 13 00037 BEQL 3$
7E D4 00039 CLRL -(SP) : 0865
08 A3 9F 0003B PUSHAB P.AAH
64 02 FB 0003E CALLS #2, DBG$PRINT
01 04 A2 CF 00041 3$: CASEL 4(SCOPE), #1, #3 : 0870
007A 0070 0024 0008 00046 4$: .WORD 5$-4$, -
6$-4$, -
8$-4$, -
10$-4$
0C AE 9F 0004E 5$: PUSHAB PATHNAME : 0880
08 A2 DD 00051 PUSHL 8(SCOPE)
02 FB 00054 CALLS #2, DBG$STA_SYMPATHNAME : 0881
10 AE 9F 00059 PUSHAB PATH_STRING
10 AE DD 0005C PUSHL PATHNAME
66 02 FB 0005F CALLS #2, DBG$NPATHDESC_TO_CS : 0882
10 AE DD 00062 PUSHL PATH_STRING
08 A3 9F 00065 PUSHAB P.AAI
51 11 00068 BRB 9$
0C A2 DD 0006A 6$: PUSHL 12(SCOPE) : 0892
0F A3 9F 0006D PUSHAB P.AAJ
64 02 FB 00070 CALLS #2, DBG$PRINT
5E DD 00073 PUSHL SP : 0893
08 AE 9F 00075 PUSHAB RSTPTR
```

RSTACCESS
V04-000

E 3
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
17-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 22
(8)

		10	AE	9F	00078	PUSHAB	MODRSTPTR	
		0C	AE	DD	0007B	PUSHL	12(SCOPE)	
0000V	CF		04	FB	0007E	CALLS	#4, DBG\$STA_NUMBERED_SCOPE	
		04	AE	D5	00083	TSTL	RSTPTR	0895
			4A	13	00086	BEQL	11\$	
			6E	D5	00088	TSTL	INVOCNUM	0898
			0E	13	0008A	BEQL	7\$	
			6E	DD	0008C	PUSHL	INVOCNUM	0900
		08	AE	DD	0008E	PUSHL	RSTPTR	
FE3D	CF		02	FB	00091	CALLS	#2, DBG\$BUILD_INVOC_RST	
04	AE		50	D0	00096	MOVL	R0, RSTPTR	
		0C	AE	9F	0009A	PUSHAB	PATHNAME	0902
		08	AE	DD	0009D	PUSHL	RSTPTR	
0000V	CF		02	FB	000A0	CALLS	#2, DBG\$STA_SYMPATHNAME	
		10	AE	9F	000A5	PUSHAB	PATH_STRING	0903
		10	AE	DD	000A8	PUSHL	PATHNAME	
	66		02	FB	000AB	CALLS	#2, DBG\$NPATHDESC_TO_LS	
		10	AE	DD	000AE	PUSHL	PATH_STRING	0904
		13	A3	9F	000B1	PUSHAB	P.AAR	
			05	11	000B4	BRB	9\$	
			7E	D4	000B6	CLRL	-(SP)	0913
		1E	A3	9F	000B8	PUSHAB	P.AAL	
	64		02	FB	000BB	CALLS	#2, DBG\$PRINT	
			12	11	000BE	BRB	11\$	
		20	A3	9F	000C0	PUSHAB	P.AAM	0920
			01	DD	000C3	PUSHL	#1	
		00028362	8F	DD	000C5	PUSHL	#164706	
00000000G	00		03	FB	000CB	CALLS	#3, LIB\$SIGNAL	
	52		62	D0	000D2	MOVL	(SCOPE), SCOPE	0927
			FF53	31	000D5	BRW	1\$	0857
00000000G	00		00	FB	000D8	CALLS	#0, DBG\$NEWLINE	0933
			04	000DF	RET			0936

; Routine Size: 224 bytes, Routine Base: DBG\$CODE + 0206

```
812 0937 1 GLOBAL ROUTINE DBG$RST_TEMP_RELEASE: NOVALUE =
813 0938 1
814 0939 1 FUNCTION
815 0940 1 This routine releases all "temporary" RST entries back to the DEBUG
816 0941 1 memory pool. "Temporary" RST entries are RST entries which are created
817 0942 1 dynamically during the execution of a DEBUG command. These include
818 0943 1 Data RST Entries for record components, RST entries for objects with
819 0944 1 invocation numbers, Line Number RST Entries, and most Data Type RST
820 0945 1 Entries. RST entries which are created by the SET MODULE command or
821 0946 1 during DEBUG initialization are not temporary RST entries.
822 0947 1
823 0948 1 When a temporary RST entry is created, it is not put on the module's
824 0949 1 symbol chain or entered in the RST Hash Table. Instead, it is added
825 0950 1 to the singly linked list pointed to by RST$TEMP_LIST. This routine
826 0951 1 is called at the end of every command to go through that list and to
827 0952 1 release every RST entry with a zero reference count to the DEBUG mem-
828 0953 1 ory pool. An entry with a non-zero reference count cannot be released
829 0954 1 since something references that entry; the current location pseudo-
830 0955 1 symbol may be bound to a Primary Descriptor which in turn points to
831 0956 1 that RST entry, for example.
832 0957 1
833 0958 1 INPUTS
834 0959 1 NONE
835 0960 1
836 0961 1 OUTPUTS
837 0962 1 NONE
838 0963 1
839 0964 1
840 0965 2 BEGIN
841 0966 2
842 0967 2 LOCAL
843 0968 2 OLDPTR: REF RST$ENTRY, ; Pointer to the previous RST entry in
844 0969 2 ; temporary RST entry list
845 0970 2 RSTPTR: REF RST$ENTRY; ; Pointer to the current RST entry in
846 0971 2 ; the temporary RST entry list
847 0972 2
848 0973 2
849 0974 2
850 0975 2 ; Loop through the Temporary RST Entry List. Release every entry with
851 0976 2 ; a zero reference count back to the memory pool.
852 0977 2
853 0978 2 OLDPTR = RST$TEMP_LIST;
854 0979 2 RSTPTR = .OLDPTR[RST$L_HASH_FLINK];
855 0980 2 WHILE .RSTPTR NEQ 0 DO
856 0981 3 BEGIN
857 0982 3 IF .RSTPTR[RST$W_REF_COUNT] EQL 0
858 0983 3 THEN
859 0984 4 BEGIN
860 0985 4 OLDPTR[RST$L_HASH_FLINK] = .RSTPTR[RST$L_HASH_FLINK];
861 0986 4 DBG$REL_MEMORY(.RSTPTR);
862 0987 4 END
863 0988 4
864 0989 3 ELSE
865 0990 3 OLDPTR = .RSTPTR;
866 0991 3
867 0992 3 RSTPTR = .OLDPTR[RST$L_HASH_FLINK];
868 0993 2 END;
```

RSTACK
V04-000

G 3
16-Sep-1984 02:48:17
14-Sep-1984 12:18:26

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]RSTACK.B32;1

Page 24
(9)

```
: 869      0994 2
: 870      0995 2
: 871      0996 2
: 872      0997 2
: 873      0998 2
: 874      0999 2
: 875      1000 1

! We are all done--return.
RETURN;
END;
```

```

53 00000000G 00 000C 00000 .ENTRY DBGSRT TEMP RELEASE, Save R2,R3
52          63 D0 00009 1$: MOVAB RSTTEMP_LIST, OLDPTR
          18 13 0000C BEQL 3$
          16 A2 B5 0000E TSTW 22(RSTPTR)
          0E 12 00011 BNEQ 2$
          63 62 D0 00013 MOVL (RSTPTR), (OLDPTR)
          52 DD 00016 PUSHL RSTPTR
00000000G 00 01 FB 00018 CALLS #1, DBGSREL_MEMORY
          E8 11 0001F BRB 1$
          53 52 D0 00021 2$: MOVL RSTPTR, OLDPTR
          E3 11 00024 BRB 1$
          04 00026 3$: RET
: 0937
: 0978
: 0979
: 0980
: 0982
: 0985
: 0986
: 0982
: 0990
: 0992
: 1000
```

; Routine Size: 39 bytes, Routine Base: DBGSRCODE + 02E6

```
877 1001 1 GLOBAL ROUTINE DBG$STA_ADDRESS_TO_REGDESCR(ADDRESS) =
878 1002 1
879 1003 1 FUNCTION
880 1004 1     This routine determines if a given address is a register address and
881 1005 1     returns a Register Descriptor if it is. If the given address points
882 1006 1     into the DBG$REG_VALUES vector, it is a register address. The register
883 1007 1     number so determined (plus a byte offset if any) and the scope number
884 1008 1     of the currently set context are combined in a 'Register Descriptor'
885 1009 1     which is then returned to the caller. (A Register Descriptor is always
886 1010 1     non-zero.) If the address is not a register, zero is returned to the
887 1011 1     caller.
888 1012 1
889 1013 1 INPUTS
890 1014 1     ADDRESS - The input address which should be checked for being a
891 1015 1     register address.
892 1016 1
893 1017 1 OUTPUTS
894 1018 1     If the given address is not a register address, zero is returned as the
895 1019 1     routine value. If it is a register address, a Register
896 1020 1     Descriptor (which is always non-zero) is returned as the
897 1021 1     routine value.
898 1022 1
899 1023 1
900 1024 2 BEGIN
901 1025 2
902 1026 2 LOCAL
903 1027 2     REGDESCR: DBG$REGDESCR;           ! Register Descriptor that we build
904 1028 2
905 1029 2
906 1030 2
907 1031 2     ! If the address is not a register address, return zero right away.
908 1032 2
909 1033 2     IF (.ADDRESS LSSA DBG$REG_VALUES[0]) OR
910 1034 3         (.ADDRESS GEQA DBG$REG_VALUES[17])
911 1035 2     THEN
912 1036 2         RETURN 0;
913 1037 2
914 1038 2
915 1039 2     ! The address is a register address. Build a Register Descriptor and
916 1040 2     ! return it to the caller.
917 1041 2
918 1042 2     REGDESCR[DBG$V_REGD_SENTINEL] = %X'2D';
919 1043 2     REGDESCR[DBG$V_REGD_OFFSET] = (.ADDRESS - DBG$REG_VALUES[0]) AND 3;
920 1044 2     REGDESCR[DBG$B_REGD_REGNUM] = (.ADDRESS - DBG$REG_VALUES[0])/%UPVAL;
921 1045 2     REGDESCR[DBG$W_REGD_SCOPENUM] = .DBG$SCOPE_NUMBER;
922 1046 2     RETURN .REGDESCR;
923 1047 2
924 1048 1 END;
```

```
53 00000000G 00 000C 00000
50          63 9E 00002
50          04 AC D1 0000C
```

```
.ENTRY DBG$STA_ADDRESS_TO_REGDESCR, Save R2,R3
MOVAB DBG$REG_VALUES, R3
MOVAB DBG$REG_VALUES, R0
CMPL ADDRESS, R0
```

```
: 1001
:
: 1033
:
```

RSTACCESS
V04-000

1 3
16-Sep-1984 02:48:17
14-Sep-1984 12:18:26

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]RSTACCESS.B32;1

Page 26
(10)

				32	1F	00010	BLSSU	1\$	
		50	44	A3	9E	00012	MOVAB	DBG\$REG_VALUES+68, R0	1034
		50	04	AC	D1	00016	CMPL	ADDRESS, R0	
				28	1E	0001A	BGEQU	1\$	
51	06	02		2D	F0	0001C	INSV	#45, #2, #6, REGDESCR	1042
		50		63	9E	00021	MOVAB	DBG\$REG_VALUES, R0	1043
		AC	04	50	C3	00024	SUBL3	R0, ADDRESS, R0	
51	02	00		50	F0	00029	INSV	R0, #0, #2, REGDESCR	
		52		04	C7	0002E	DIVL3	#4, R0, R2	1044
51	08	08		52	F0	00032	INSV	R2, #8, #8, REGDESCR	
51	10	10	00000000'	EF	F0	00037	INSV	DBG\$SCOPE_NUMBER, #16, #16, REGDESCR	1045
		50		51	D0	00040	MOVL	REGDESCR, R0	1046
					04	00043	RET		
				50	D4	00044	CLRL	R0	1048
					04	00046	RET		

; Routine Size: 71 bytes, Routine Base: DBG\$CODE + 030D


```

926 1049 1 GLOBAL ROUTINE DBG$STA_GETSOURCEMOD(MODNAMEPTR) =
927 1050 1
928 1051 1 FUNCTION
929 1052 1 This routine looks up what module should be used when displaying source
930 1053 1 lines. It accepts a pointer to a Counted ASCII module name and returns
931 1054 1 a pointer to the corresponding Module RST Entry. However, if the name
932 1055 1 pointer is zero, it determines which module contains the current scope
933 1056 1 (as defined by the Scope List) and returns a pointer to that module's
934 1057 1 Module RST Entry. If the module name does not exist or if no known
935 1058 1 module contains the current scope, the routine returns a zero value.
936 1059 1
937 1060 1 This routine is called during the processing of the TYPE command to
938 1061 1 determine which module to type source lines from. It is also called
939 1062 1 during the processing of the SET SOURCE/MODULE and CANCEL SOURCE/MODULE
940 1063 1 commands to look up module names. Only the TYPE command passes a zero
941 1064 1 module name pointer; this happens when no module name is specified on
942 1065 1 the command.
943 1066 1
944 1067 1 INPUTS
945 1068 1 MODNAMEPTR - A pointer to the Counted ASCII module name to be looked up.
946 1069 1 If the module of the current scope is to be looked up, this
947 1070 1 pointer should be zero.
948 1071 1
949 1072 1 OUTPUTS
950 1073 1 A pointer to the Module RST Entry of the module specified by MODNAMEPTR
951 1074 1 is returned as the routine value. If the desired module could
952 1075 1 not be found (no such module name or current scope not in any
953 1076 1 known module), zero is returned as the value.
954 1077 1
955 1078 1
956 1079 2 BEGIN
957 1080 2
958 1081 2 LOCAL
959 1082 2 INVOCNUM, ! Invocation number parameter
960 1083 2 MODRSTPTR: REF RST$ENTRY, ! Pointer to found Module RST Entry
961 1084 2 SCOPE, ! Scope pointer parameter
962 1085 2 SCOPEPTR: REF SCOPE$ENTRY; ! Pointer to current Scope List Entry
963 1086 2
964 1087 2
965 1088 2
966 1089 2 ! If the MODNAMEPTR parameter is non-zero, we search the RST Hash Table for
967 1090 2 ! the Counted ASCII module name pointed to by MODNAMEPTR.
968 1091 2
969 1092 2 IF .MODNAMEPTR NEQ 0
970 1093 2 THEN
971 1094 3 BEGIN
972 1095 3 DBG$HASH FIND_SETUP(.MODNAMEPTR);
973 1096 3 WHILE TRUE DO
974 1097 4 BEGIN
975 1098 4 MODRSTPTR = DBG$HASH FIND(.MODNAMEPTR);
976 1099 4 IF .MODRSTPTR EQL 0 THEN RETURN 0;
977 1100 4 IF .MODRSTPTR[RST$B_KIND] EQL RST$K_MODULE THEN RETURN .MODRSTPTR;
978 1101 3 END;
979 1102 3
980 1103 2 END;
981 1104 2
982 1105 2
```

```

: 983      1106 2      ! MODNAMEPTR is zero. We thus search the Scope List for the first scope
: 984      1107 2      ! with a known module and return a pointer to that Module RST Entry.
: 985      1108 2
: 986      1109 2      SCOPEPTR = .SCOPE$LIST;
: 987      1110 2      WHILE .SCOPEPTR NEQ 0 DO
: 988      1111 2          BEGIN
: 989      1112 2              CASE .SCOPEPTR[SCOPE$L_STATE] FROM SCOPE$K_NORMAL TO SCOPE$K_SETMODS OF
: 990      1113 2                  SET
: 991      1114 2
: 992      1115 2
: 993      1116 2              ! Normal scope--just return the scope's Module RST Entry pointer.
: 994      1117 2              [SCOPE$K_NORMAL]:
: 995      1118 2                  RETURN .SCOPEPTR[SCOPE$L_MODPTR];
: 996      1119 2
: 997      1120 2
: 998      1121 2              ! Numbered scope--look up the corresponding lexical entity and
: 999      1122 2              ! module in the call stack and return its Module RST Entry pointer.
1000      1123 2              ! If the module is not found, we continue the Scope List search.
1001      1124 2              [SCOPE$K_NUMBERED]:
1002      1125 2                  BEGIN
1003      1126 2                      DBG$STA_NUMBERED_SCOPE(.SCOPEPTR[SCOPE$L_MODPTR], MODRSTPTR,
1004      1127 2                      SCOPE, INVOCNUM);
1005      1128 2                      IF .MODRSTPTR NEQ 0 THEN RETURN .MODRSTPTR;
1006      1129 2                      END;
1007      1130 2
1008      1131 2
1009      1132 2              ! Global symbol scope--just ignore this entry and continue search.
1010      1133 2              [SCOPE$K_GLOBAL]:
1011      1134 2                  0;
1012      1135 2
1013      1136 2
1014      1137 2
1015      1138 2
1016      1139 2
1017      1140 2              ! All SET modules scope--ignore this entry and continue search.
1018      1141 2              [SCOPE$K_SETMODS]:
1019      1142 2                  0;
1020      1143 2
1021      1144 2
1022      1145 2              TES;
1023      1146 2
1024      1147 2
1025      1148 2              ! Link to the next Scope List Entry and loop.
1026      1149 2              SCOPEPTR = .SCOPEPTR[SCOPE$L_FLINK];
1027      1150 2              END;
1028      1151 2
1029      1152 2
1030      1153 2
1031      1154 2      ! No usable scope was found in the Scope List. Return zero to the caller.
1032      1155 2      RETURN 0;
1033      1156 2
1034      1157 2
1035      1158 1      END;
```

				0004	00000		.ENTRY	DBG\$STA_GETSOURCEMOD, Save R2	:	1049
	5E			0C	C2	00002	SUBL2	#12, SP	:	
	52		04	AC	D0	00005	MOVL	MODNAMEPTR, R2	:	1092
				1F	13	00009	BEQL	2\$:	
				52	DD	0000B	PUSHL	R2	:	1095
	00000000G	00		01	FB	0000D	CALLS	#1, DBG\$HASH_FIND_SETUP	:	
				52	DD	00014	PUSHL	R2	:	1098
	00000000G	00		01	FB	00016	CALLS	#1, DBG\$HASH_FIND	:	
	08	AE		50	D0	0001D	MOVL	R0, MODRSTPTR	:	
				41	13	00021	BEQL	8\$:	1099
		01	14	A0	91	00023	CMPB	20(R0), #1	:	1100
				EB	12	00027	BNEQ	1\$:	
					04	00029	RET		:	
		52	00000000G	00	D0	0002A	MOVL	SCOPE\$LIST, SCOPEPTR	:	1109
				31	13	00031	BEQL	8\$:	1110
		01		A2	CF	00033	CASEL	4(SCOPEPTR), #1, #3	:	1112
0027	0027	000D		0008		00038	.WORD	5\$-4\$,- 6\$-4\$,- 7\$-4\$,- 7\$-4\$:	
									:	
		50		0C	A2	D0	MOVL	12(SCOPEPTR), R0	:	1119
					04	00044	RET		:	
				5E	DD	00045	PUSHL	SP	:	1128
			08	AE	9F	00047	PUSHAB	SCOPE	:	
			10	AE	9F	0004A	PUSHAB	MODRSTPTR	:	
			0C	A2	DD	0004D	PUSHL	12(SCOPEPTR)	:	
	0000V	CF		04	FB	00050	CALLS	#4, DBG\$STA_NUMBERED_SCOPE	:	
			08	AE	D5	00055	TSTL	MODRSTPTR	:	1130
				05	13	00058	BEQL	7\$:	
		50		08	AE	D0	MOVL	MODRSTPTR, R0	:	
					04	0005E	RET		:	
		52			62	D0	MOVL	(SCOPEPTR), SCOPEPTR	:	1150
					CD	11	BRB	3\$:	1110
					50	D4	CLRL	R0	:	1158
					04	00066	RET		:	

; Routine Size: 103 bytes, Routine Base: DBG\$CODE + 0354

```
1037 1159 1 GLOBAL ROUTINE DBG$STA GETSYMBOL(PATHNAME, SYMID, KIND,  
1038 1160 1 OUT_SCOPE_STATE, OUT_SCOPE, ARRAY_FLAG,  
1039 1161 1 TYPE_FLAG): NOVALUE =  
1040 1162 1  
1041 1163 1 FUNCTION:  
1042 1164 1 This routine accepts a pathname and returns the corresponding symbol.  
1043 1165 1 The pathname, which is passed in internal format, consists of a symbolic  
1044 1166 1 name (such as 'X') or a symbolic name with pathname qualification (such  
1045 1167 1 as 'M\R\X'). It also includes any data record qualification which may  
1046 1168 1 be present; thus 'M\R\A.B.C' constitutes a pathname in this context.  
1047 1169 1 This routine is the central routine one calls to search the Debug Symbol  
1048 1170 1 Table (the DST) to find the symbol table entry corresponding to a given  
1049 1171 1 symbolic name. The search takes into account all scope rules dictated  
1050 1172 1 by the language and the SET SCOPE and SET MODULE commands.  
1051 1173 1  
1052 1174 1 INPUTS:  
1053 1175 1 PATHNAME - The address of a pathname descriptor describing the symbolic  
1054 1176 1 name to be looked up in the DST. A 'pathname descriptor' is  
1055 1177 1 the internal data structure which describes an already parsed  
1056 1178 1 symbolic name including all pathname and data record qualifi-  
1057 1179 1 cation.  
1058 1180 1  
1059 1181 1 SYMID - The address of a longword location where the 'symbol identi-  
1060 1182 1 fier' should be returned. The 'symbol identifier' is a value  
1061 1183 1 which uniquely identifies the returned symbol. This value is  
1062 1184 1 not directly understood outside the symbol table access rou-  
1063 1185 1 tines, but can be passed to various other symbol table access  
1064 1186 1 routines to extract information about the symbol.  
1065 1187 1  
1066 1188 1 KIND - The address of a longword location where the 'kind' of the  
1067 1189 1 SYMID symbol should be returned. KIND specifies whether SYMID  
1068 1190 1 identifies a routine, a line, or a data item, etc. See the  
1069 1191 1 OUTPUTS section below for more detail.  
1070 1192 1  
1071 1193 1 OUT_SCOPE_STATE - If not zero, the caller wishes to have  
1072 1194 1 passed back to him the scope state  
1073 1195 1 (e.g., NORMAL, SETMODS, ...) in which  
1074 1196 1 the lookup succeeded.  
1075 1197 1  
1076 1198 1 OUT_SCOPE - If not zero, the caller wishes to have passed  
1077 1199 1 back to him the scope in which the lookup  
1078 1200 1 of the symbol succeeded.  
1079 1201 1  
1080 1202 1 ARRAY_FLAG - Indicates that this symbol lookup was  
1081 1203 1 called as part of processing a subscripted symbol.  
1082 1204 1 This information is used in BASIC to disambiguate  
1083 1205 1 symbol references. That is, in BASIC, you can have  
1084 1206 1 2 symbols named X, one an array and one not, and  
1085 1207 1 the language uses context to tell them apart.  
1086 1208 1  
1087 1209 1 TYPE_FLAG - Indicates that it is OK to return a RSTPTR whose  
1088 1210 1 kind is 'TYPE'.  
1089 1211 1  
1090 1212 1 OUTPUTS:  
1091 1213 1 SYMID - A symbol identifier which uniquely identifies the symbol spec-  
1092 1214 1 ified by PATHNAME is returned to SYMID. This symbol identi-  
1093 1215 1 fier can then be passed to any symbol table access routine
```

```

: 1094 1216 1
: 1095 1217 1
: 1096 1218 1
: 1097 1219 1
: 1098 1220 1
: 1099 1221 1
: 1100 1222 1
: 1101 1223 1
: 1102 1224 1
: 1103 1225 1
: 1104 1226 1
: 1105 1227 1
: 1106 1228 1
: 1107 1229 1
: 1108 1230 1
: 1109 1231 1
: 1110 1232 1
: 1111 1233 1
: 1112 1234 1
: 1113 1235 1
: 1114 1236 2
: 1115 1237 2
: 1116 1238 2
: 1117 1239 2
: 1118 1240 2
: 1119 1241 2
: 1120 1242 2
: 1121 1243 2
: 1122 1244 2
: 1123 1245 2
: 1124 1246 2
: 1125 1247 2
: 1126 1248 2
: 1127 1249 2
: 1128 1250 2
: 1129 1251 2
: 1130 1252 2
: 1131 1253 2
: 1132 1254 2
: 1133 1255 2
: 1134 1256 2
: 1135 1257 2
: 1136 1258 2
: 1137 1259 2
: 1138 1260 2
: 1139 1261 2
: 1140 1262 2
: 1141 1263 2
: 1142 1264 2
: 1143 1265 2
: 1144 1266 2
: 1145 1267 2
: 1146 1268 2
: 1147 1269 2
: 1148 1270 2
: 1149 1271 2
: 1150 1272 2

```

which accepts a SYMID parameter. If no unique symbol corresponding to PATHNAME can be found in the DST (given the scope rules in effect), a zero is returned to SYMID.

KIND - The 'kind' of the SYMID symbol is returned to KIND. This is a small integer value with the following possible values:

```

RST$K_INVALID -- No symbol was found (SYMID = 0)
RST$K_NOTUNIQUE -- Symbol is not unique (SYMID = 0)
RST$K_ROUTINE -- SYMID is a Routine
RST$K_BLOCK -- SYMID is a Block
RST$K_ENTRY -- SYMID is an Entry Point
RST$K_LABEL -- SYMID is a Label
RST$K_LINE -- SYMID is a Line
RST$K_DATA -- SYMID is a Data Item
RST$K_TYPCOMP -- SYMID is a Data Type Component

```

No value is returned by DBG\$STA_GETSYMBOL.

BEGIN

MAP

```

PATHNAME: REF PTH$PATHNAME, : Pointer to input pathname descriptor
SYMID: REF VECTOR[1], : Pointer to SYMID location
KIND: REF VECTOR[1]; : Pointer to KIND location

```

LITERAL

```

MAX_STACK = 100; : Maximum size of the symbol name stack

```

FIELD STK_FLDS =

```

SET : Field definitions for SYMSTACK vector
STK_RSTPTR = [ 0, L ], : RST pointer to current stack component
STK_PINDEX = [ 1, W0 ], : Pathname vector index + 1
STK_TPINDEX = [ 1, W1 ], : Next Type RST Entry ref table index
TES;

```

OWN

```

CANDLIST: REF VECTOR[.LONG] : Pointer to RST entry candidate list
INITIAL(0), : for the current scope
MODU_SCOPE: SCOPE$ENTRY : Scope list entry used for explicitly
INITIAL(0, SCOPE$K_NORMAL, 0, 0), : specified module scope
NORM_SCOPE: SCOPE$ENTRY : Scope list entry used for explicitly
INITIAL(0, SCOPE$K_NORMAL, 0, 0), : specified scopes
NUMB_SCOPE: SCOPE$ENTRY : Scope list entry used for numbered
INITIAL(0, SCOPE$K_NUMBERED, 0, 0); : scopes (i.e., n\X).

```

LOCAL

```

ARR_FLAG, : TRUE if symbol is subscripted
CANDBLK: REF CAND_BLOCKVECTOR, : Pointer to candidate block-vector
DEFDEPTH, : Definition depth of symbol in scope
FIRST_MODPTR, : Pointer to first module on scope list
GOOD_CAND, : CANDLIST index of good candidate symbol
HAVE_LINE_NUM, : flag set if pathname has a line number
HAVE_NUM_SCOPE, : flag set if pathname has a numbered
: scope in first position ('0\1')
HAVE_SCOPE, : flag set when we have scope to search

```

1151	1273	2	IN_SCOPE,	Flag set if symbol is in current scope
1152	1274	2	J	Loop index for CANDBLK vector
1153	1275	2	LINEEND,	Line number end address
1154	1276	2	LINE_LEX_PTR,	Pointer to the inner-most lexical
1155	1277	2		entity containing the line number
1156	1278	2	LINE_NUM,	Line number if pathname contains one
1157	1279	2	LINE_NUM_IS_LAST,	Flag set if there is a line number and
1158	1280	2		it is last in the pathname
1159	1281	2	LINE_NUM_LOC,	Index of line number (if present) in
1160	1282	2		pathname vector (1-based).
1161	1283	2	LINESTART,	Line number start address
1162	1284	2	MODRSTPTR: REF RST\$ENTRY,	Pointer to Module RST Entry for the
1163	1285	2		current scope being searched
1164	1286	2	NAMEPTR: REF VECTOR[.BYTE],	Pointer to RST entry symbol name as
1165	1287	2		a counted ASCII string
1166	1288	2	NCANDS,	Number of candidate list entries
1167	1289	2	NEWREFLIST,	Temporary pointer to new RST Reference
1168	1290	2		List memory block
1169	1291	2	NEXTSETMOD: REF RST\$ENTRY,	Pointer to the next SET module after
1170	1292	2		this one--used when searching all
1171	1293	2		SET modules for a pathname match
1172	1294	2	NUMBER,	Used to convert line number to binary
1173	1295	2	NUMSCP_INVOC_NUM,	Invocation number of the current
1174	1296	2		numbered scope
1175	1297	2	OLDCAND,	Pointer to candidate list about to be
1176	1298	2		copied to a larger memory block
1177	1299	2	PATH_NAME_PTR,	Pointer to pathname counted ASCII
1178	1300	2	PATH_START_LOC,	Start location of scope in pathname
1179	1301	2	PATHSTRING,	Pointer to pathname counted ASCII
1180	1302	2	PATHVEC: REF VECTOR[.LONG],	Pointer to the pathname vector
1181	1303	2	PINDEX,	Index into pathname vector
1182	1304	2	PNAME: REF VECTOR[.BYTE],	Pointer to pathname component counted
1183	1305	2		ASCII string
1184	1306	2	REGISTER_SYMID: REF RST\$ENTRY,	SYMID for register name (such as %R5)
1185	1307	2	REG_LINE_LEX_PTR,	Same as LINE_LEX_PTR but for the scope
1186	1308	2		in which registers are looked up
1187	1309	2	REG_SCOPE,	Set to TRUE if current scope is scope
1188	1310	2		in which registers are looked up
1189	1311	2	RNAME: REF VECTOR[.BYTE],	Pointer to RST entry scope chain com-
1190	1312	2		ponent's name as counted ASCII
1191	1313	2	ROUTPTR: REF RST\$ENTRY,	Pointer to Routine RST Entry of rout-
1192	1314	2		ine with invocation number
1193	1315	2	RPTR: REF RST\$ENTRY,	Pointer to current RST entry in RSTPTR
1194	1316	2		entry's up-scope chain
1195	1317	2	RSTPTR: REF RST\$ENTRY,	Pointer to candidate RST entry
1196	1318	2	SATPTR: REF SAT\$ENTRY,	Pointer to Static Address Table entry
1197	1319	2	SCOPE: REF RST\$ENTRY,	Pointer to current scope's RST entry
1198	1320	2	SCOPEPTR: REF SCOPE\$ENTRY,	Points to the current scope list entry
1199	1321	2	SCOPE_START_PTR,	Pointer to start of scope list actual-
1200	1322	2		ly searched--used for registers
1201	1323	2	SCOPE_STATE,	The current state in our traversal of
1202	1324	2		the scopes to be searched
1203	1325	2	SCPTR: REF RST\$ENTRY,	Pointer used to follow current scope's
1204	1326	2		up-scope chain
1205	1327	2	SET_SCOPE,	Set to TRUE if called by SET SCOPE
1206	1328	2	STATUS,	Status code returned by called routine
1207	1329	2	STKPTR,	Current SYMSTACK index

```
1208 1330 2      STMT_NUM,           Statement number within line number
1209 1331 2      SYMSCOPE: REF RST$ENTRY, The actual scope of the current symbol
1210 1332 2      SYMSTACK:       Symbol name stack for name components
1211 1333 2      BLOCKVECTOR[MAX_STACK, 2] LONG] FIELD(STK_FLDS), !
1212 1334 2      TPINDEX,       Index into a Type RST Entry's table of
1213 1335 2      TPTR: REF VECTOR[,LONG], references to that type
1214 1336 2      VALID_LINE_FLAG; Pointer to the Type RST Entry refer-
1215 1337 2      ence table
1216 1338 2      Flag set if line number is valid
1217 1339 2
1218 1340 2
1219 1341 2
1220 1342 2      ! Note whether we were called by the SET scope command.
1221 1343 2
1222 1344 2      SET SCOPE = .RST$SET SCOPE;
1223 1345 2      RST$SET_SCOPE = FALSE;
1224 1346 2
1225 1347 2
1226 1348 2      ! Set up pointers to the pathname descriptor's pathname vector and the last
1227 1349 2      name in the pathname vector.
1228 1350 2
1229 1351 2      PATHVEC = PATHNAME[PTHS$A PATHVECTOR];
1230 1352 2      NAMEPTR = .PATHVEC[.PATHNAME[PTHS$B_TOTCNT] - 1];
1231 1353 2
1232 1354 2
1233 1355 2      ! See if there is any line number reference in the pathname. If there is,
1234 1356 2      extract the line and statement numbers and set the HAVE_LINE_NUM flag.
1235 1357 2
1236 1358 2      HAVE_LINE_NUM = FALSE;
1237 1359 2      LINE_NUM_IS_LAST = FALSE;
1238 1360 2      LINE_NUM_LOC = 0;
1239 1361 2      VALID_LINE_FLAG = TRUE;
1240 1362 2      INCR I FROM 1 TO .PATHNAME[PTHS$B_TOTCNT] DO
1241 1363 3      BEGIN
1242 1364 3      PNAME = .PATHVEC[.I - 1];
1243 1365 3      IF (.PNAME[0] GTR 6) AND
1244 1366 3      CH$EQL(6, PNAME[1], 6, UPLIT BYTE(%ASCII '%LINE '), 0)
1245 1367 3      THEN
1246 1368 4      BEGIN
1247 1369 4      IF .PNAME[7] LSS '0' OR .PNAME[7] GTR '9' THEN VALID_LINE_FLAG = FALSE;
1248 1370 4      IF .HAVE_LINE_NUM THEN VALID_LINE_FLAG = FALSE;
1249 1371 4      HAVE_LINE_NUM = TRUE;
1250 1372 4      LINE_NUM_LOC = .I;
1251 1373 4
1252 1374 4
1253 1375 4      ! Loop over the line number ASCII to pick up the actual line number
1254 1376 4      and statement number.
1255 1377 4
1256 1378 4      LINE_NUM = -1;
1257 1379 4      NUMBER = 0;
1258 1380 4      INCR I FROM 7 TO .PNAME[0] DO
1259 1381 5      BEGIN
1260 1382 5      IF .PNAME[.I] EQL '.' AND .LINE_NUM EQL -1
1261 1383 5      THEN
1262 1384 6      BEGIN
1263 1385 6      LINE_NUM = .NUMBER;
1264 1386 6      NUMBER = 0;
```

```
1265 1387 6      END
1266 1388 6
1267 1389 5      ELSE IF (.PNAME[.I] GEQ '0') AND (.PNAME[.I] LEQ '9') AND
1268 1390 6          (.NUMBER LEQ 1000000)
1269 1391 5      THEN
1270 1392 6          NUMBER = 10*.NUMBER + (.PNAME[.I] - '0')
1271 1393 6
1272 1394 5      ELSE
1273 1395 6          BEGIN
1274 1396 6          VALID_LINE_FLAG = FALSE;
1275 1397 6          EXITLOOP;
1276 1398 5          END;
1277 1399 5
1278 1400 4      END;
1279 1401 4
1280 1402 4
1281 1403 4      ! Set LINE_NUM and STMT_NUM properly on loop exit.
1282 1404 4      !
1283 1405 4      IF .LINE_NUM EQL -1
1284 1406 4      THEN
1285 1407 5          BEGIN
1286 1408 5          LINE_NUM = .NUMBER;
1287 1409 5          STMT_NUM = 0;
1288 1410 5          END
1289 1411 5
1290 1412 4      ELSE
1291 1413 4          STMT_NUM = .NUMBER;
1292 1414 4
1293 1415 3      END;
1294 1416 3
1295 1417 2      END;
1296 1418 2          ! End of line number INCR loop
1297 1419 2
1298 1420 2      ! If we got a line number, make some additional validity checks on it.
1299 1421 2      ! If the line number is not valid for any reason (including syntax errors),
1300 1422 2      ! return the invalid symbol code to the caller.
1301 1423 2      !
1302 1424 2      IF .HAVE_LINE_NUM
1303 1425 2      THEN
1304 1426 3          BEGIN
1305 1427 3          LINE_NUM_IS_LAST = .LINE_NUM LOC EQL .PATHNAME[PTHSB_TOTCNT];
1306 1428 3          IF (.LINE_NUM LOC GTR .PATHNAME[PTHSB_PATHCNT]) OR
1307 1429 3              (.LINE_NUM LOC LSS .PATHNAME[PTHSB_PATHCNT] - 1) OR
1308 1430 4              (.LINE_NUM LOC EQL .PATHNAME[PTHSB_PATHCNT] AND NOT .LINE_NUM_IS_LAST)
1309 1431 3          THEN
1310 1432 3              VALID_LINE_FLAG = FALSE;
1311 1433 3
1312 1434 3          IF NOT .VALID_LINE_FLAG
1313 1435 3          THEN
1314 1436 4              BEGIN
1315 1437 4              SYMID[0] = 0;
1316 1438 4              KIND[0] = RST$K_INVALID;
1317 1439 4              RETURN;
1318 1440 3              END;
1319 1441 3
1320 1442 2      END;
1321 1443 2
```



```
1322 1444 2
1323 1445 2
1324 1446 2
1325 1447 2
1326 1448 2
1327 1449 2
1328 1450 2
1329 1451 2
1330 1452 2
1331 1453 2
1332 1454 2
1333 1455 2
1334 1456 2
1335 1457 2
1336 1458 2
1337 1459 2
1338 1460 2
1339 1461 2
1340 1462 2
1341 1463 2
1342 1464 2
1343 1465 2
1344 1466 2
1345 1467 2
1346 1468 2
1347 1469 2
1348 1470 2
1349 1471 2
1350 1472 2
1351 1473 2
1352 1474 2
1353 1475 2
1354 1476 2
1355 1477 2
1356 1478 2
1357 1479 2
1358 1480 2
1359 1481 2
1360 1482 2
1361 1483 2
1362 1484 2
1363 1485 2
1364 1486 2
1365 1487 2
1366 1488 2
1367 1489 2
1368 1490 2
1369 1491 2
1370 1492 2
1371 1493 2
1372 1494 2
1373 1495 2
1374 1496 2
1375 1497 2
1376 1498 2
1377 1499 2
1378 1500 2

! If we do not yet have a candidate list memory block, get one and initial-
! ize its first element to give the list size that will fit in the block.
NCANDS = 0;
IF .CANDLST EQL 0
THEN
  BEGIN
    CANDLST = DBG$GET_MEMORY(11);
    CANDLST[0] = 10;
  END;

! Set up the "scope pointer" to point to the list of scopes to be searched.
! If the symbol is of the form \X, we search the Global Symbol Table only,
! and if it is of the form n\X, we search the n-th "numbered scope" only.
! Otherwise, we use the normal scope list given by SCOPE$LIST.
SCOPEPTR = .SCOPE$LIST;
HAVE_NUM_SCOPE = FALSE;
PNAME = .PATHVEC[0];
IF .PNAME[0] EQL 0
THEN
  BEGIN
    IF .PATHNAME[PTH$B_LOCINVO] EQL 0
    THEN
      SCOPEPTR = UPLIT(0, SCOPE$K_GLOBAL, 0, 0)

    ELSE IF .PATHNAME[PTH$B_LOCINVO] EQL 1
    THEN
      BEGIN
        HAVE_NUM_SCOPE = TRUE;
        SCOPEPTR = NUMB_SCOPE;
        SCOPEPTR[SCOPE$C_MODPTR] = .PATHNAME[PTH$B_INVOCNUM];
        IF .PATHNAME[PTH$B_PATHCNT] LSS 2 THEN SCOPEPTR = 0;
      END

    ELSE
      $DBG_ERROR('RSTACCESS\GETSYMBOL');

  END

! If there is pathname qualification on the variable name other than the
! global scope or a numbered scope, we determine what scope is specified
! and set up a scope list entry for that scope.
ELSE IF (.PATHNAME[PTH$B_PATHCNT] GTR 1) AND (.LINE_NUM_LOC NEQ 1)
THEN
  BEGIN
    PATH_START_LOC = .PATHNAME[PTH$B_PATHCNT] - 1;
    IF .LINE_NUM_LOC EQL .PATH_START_LOC
    THEN
      PATH_START_LOC = .PATH_START_LOC - 1;

! Loop over the RST Hash Table chain for this symbol name (i.e., for the
```

```

: 1379      1501      3      ! scope specified by the pathname). For each such symbol, see if it
: 1380      1502      3      ! matches the rest of the pathname qualification.
: 1381      1503      3
: 1382      1504      3      MODU_SCOPE[SCOPE$L_RSTPTR] = 0;
: 1383      1505      3      NORM_SCOPE[SCOPE$L_RSTPTR] = 0;
: 1384      1506      3      PATH_NAME_PTR = .PATHVEC[.PATH_START_LOC - 1];
: 1385      1507      3      DBG$HASH_FIND_SETUP(.PATH_NAME_PTR);
: 1386      1508      3      WHILE TRUE DO
: 1387      1509      4          BEGIN
: 1388      1510      4
: 1389      1511      4          ! Get the next symbol from the hash chain. Check for chain's end.
: 1390      1512      4          !
: 1391      1513      4          RSTPTR = DBG$HASH_FIND(.PATH_NAME_PTR);
: 1392      1514      4          IF .RSTPTR EQL 0 THEN EXITLOOP;
: 1393      1515      4
: 1394      1516      4          ! Scan the symbol's up-scope chain and see if it fully matches the
: 1395      1517      4          ! specified pathname qualification. If so, this is the scope.
: 1396      1518      4          !
: 1397      1519      4          RPTR = .RSTPTR;
: 1398      1520      4          PINDEX = .PATH_START_LOC;
: 1399      1521      4          WHILE TRUE DO
: 1400      1522      4              BEGIN
: 1401      1523      4                  IF .RSTPTR[RST$V_GLOBAL] THEN EXITLOOP;
: 1402      1524      5                  IF (.RSTPTR[RST$B_KIND] NEQ RST$K_MODULE) AND
: 1403      1525      5                      (.RSTPTR[RST$B_KIND] NEQ RST$K_ROUTINE) AND
: 1404      1526      5                      (.RSTPTR[RST$B_KIND] NEQ RST$K_BLOCK) AND
: 1405      1527      5                      (.RSTPTR[RST$B_KIND] NEQ RST$K_TYPE)
: 1406      1528      5                  THEN
: 1407      1529      6                      EXITLOOP;
: 1408      1530      5
: 1409      1531      5                  PNAME = .PATHVEC[.PINDEX - 1];
: 1410      1532      5                  RNAME = DBG$GET_DST_NAME(.RPTR[RST$L_DSTPTR]);
: 1411      1533      5                  IF CH$EQL(.PNAME[0], PNAME[1], .RNAME[0], RNAME[1], 0)
: 1412      1534      5                  THEN
: 1413      1535      5                      BEGIN
: 1414      1536      5                          ! If the pathname ended and everything matched so far, this
: 1415      1537      6                          ! may be the desired scope. Create a scope list entry for
: 1416      1538      6                          ! it and check for uniqueness.
: 1417      1539      6                          !
: 1418      1540      6                          IF .PINDEX EQL 1
: 1419      1541      6                          THEN
: 1420      1542      6                              BEGIN
: 1421      1543      6                                  MODRSTPTR = .RSTPTR;
: 1422      1544      6                                  WHILE .MODRSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
: 1423      1545      6                                      MODRSTPTR = .MODRSTPTR[RST$L_UPSCOPEPTR];
: 1424      1546      7
: 1425      1547      7                                  SCOPEPTR = NORM_SCOPE;
: 1426      1548      7                                  IF .RSTPTR EQL .MODRSTPTR THEN SCOPEPTR = MODU_SCOPE;
: 1427      1549      7                                  IF .SCOPEPTR[SCOPE$L_RSTPTR] NEQ 0
: 1428      1550      7                                  THEN
: 1429      1551      7                                      BEGIN
: 1430      1552      7                                          SYMID[0] = 0;
: 1431      1553      7                                          KIND[0] = RST$K_NOTUNIQUE;
: 1432      1554      7
: 1433      1555      8
: 1434      1556      8
: 1435      1557      8
```

```

: 1436      1558      8
: 1437      1559      7
: 1438      1560      7
: 1439      1561      7
: 1440      1562      7
: 1441      1563      7
: 1442      1564      6
: 1443      1565      6
: 1444      1566      6
: 1445      1567      6
: 1446      1568      6
: 1447      1569      6
: 1448      1570      5
: 1449      1571      5
: 1450      1572      5
: 1451      1573      5
: 1452      1574      5
: 1453      1575      5
: 1454      1576      5
: 1455      1577      4
: 1456      1578      4
: 1457      1579      3
: 1458      1580      3
: 1459      1581      3
: 1460      1582      3
: 1461      1583      3
: 1462      1584      3
: 1463      1585      3
: 1464      1586      3
: 1465      1587      3
: 1466      1588      3
: 1467      1589      3
: 1468      1590      4
: 1469      1591      4
: 1470      1592      4
: 1471      1593      3
: 1472      1594      3
: 1473      1595      3
: 1474      1596      2
: 1475      1597      2
: 1476      1598      2
: 1477      1599      2
: 1478      1600      2
: 1479      1601      2
: 1480      1602      2
: 1481      1603      2
: 1482      1604      2
: 1483      1605      2
: 1484      1606      2
: 1485      1607      3
: 1486      1608      3
: 1487      1609      3
: 1488      1610      2
: 1489      1611      2
: 1490      1612      2
: 1491      1613      2
: 1492      1614      2

      RETURN;
      END;

      SCOPEPTR[SCOPE$L_RSTPTR] = .RSTPTR;
      SCOPEPTR[SCOPE$L_MODPTR] = .MODRSTPTR;
      EXITLOOP;
      END;

      ! Decrement the PATHVEC index and continue the search.
      !
      PINDEX = .PINDEX - 1;
      END;

      ! Link up-scope and continue the search.
      !
      IF .RPTR[RST$B_KIND] EQL RST$K_MODULE THEN EXITLOOP;
      RPTR = .RPTR[RST$L_UPSCOPEPTR];
      END;

      END;                                ! End of WHILE loop over hash table

      ! Depending on whether a normal scope or a module scope or both were
      ! found to match the pathname, put the corresponding scope list entries
      ! on the scope list.
      SCOPEPTR = 0;
      MODU_SCOPE[SCOPE$L_FLINK] = 0;
      IF .NORM_SCOPE[SCOPE$L_RSTPTR] NEQ 0
      THEN
      BEGIN
      MODU_SCOPE[SCOPE$L_FLINK] = NORM_SCOPE;
      SCOPEPTR = NORM_SCOPE;
      END;

      IF .MODU_SCOPE[SCOPE$L_RSTPTR] NEQ 0 THEN SCOPEPTR = MODU_SCOPE;
      END;

      ! Set up NEXTSETMOD for a search through all SET modules. Also save the
      ! scope-list start pointer so we can look up the symbol in that scope in
      ! case it happens to be register name.
      NEXTSETMOD = .RST$START_ADDR;
      REG_SCOPE = FALSE;
      REG_LINE_LEX_PTR = 0;
      FIRST_MODPTR = 0;
      SCOPE_START_PTR = .SCOPEPTR;
      IF (.SCOPEPTR EQL MODU_SCOPE) AND (.MODU_SCOPE[SCOPE$L_FLINK] NEQ 0)
      THEN
      SCOPE_START_PTR = .MODU_SCOPE[SCOPE$L_FLINK];

      ! Loop through all the proper scopes, searching for a symbol which matches
      ! the specified pathname.
```

```
1493 1615 2
1494 1616 2
1495 1617 2
1496 1618 2
1497 1619 2
1498 1620 2
1499 1621 2
1500 1622 2
1501 1623 2
1502 1624 3
1503 1625 4
1504 1626 4
1505 1627 4
1506 1628 4
1507 1629 4
1508 1630 4
1509 1631 4
1510 1632 4
1511 1633 4
1512 1634 4
1513 1635 4
1514 1636 4
1515 1637 5
1516 1638 5
1517 1639 5
1518 1640 5
1519 1641 5
1520 1642 5
1521 1643 5
1522 1644 5
1523 1645 5
1524 1646 5
1525 1647 5
1526 1648 5
1527 1649 5
1528 1650 5
1529 1651 5
1530 1652 5
1531 1653 6
1532 1654 7
1533 1655 6
1534 1656 6
1535 1657 6
1536 1658 6
1537 1659 6
1538 1660 6
1539 1661 6
1540 1662 5
1541 1663 5
1542 1664 5
1543 1665 5
1544 1666 5
1545 1667 5
1546 1668 5
1547 1669 5
1548 1670 5
1549 1671 5

!
WHILE TRUE DO
  BEGIN

    ! Loop through the scope selection code until we find a scope in which
    ! to search for the specified pathname.

    HAVE_SCOPE = FALSE;
    WHILE TRUE DO
      BEGIN

        ! If the scope list has no more entries, we have searched all scopes
        ! on the list without finding the desired symbol. This means that
        ! the symbol is not in the RST so we return RST$K_INVALID as the
        ! status. However, we first call GET_REGISTER_SYMID to see if the
        ! symbol could be a register name (e.g., 'R5'). If so, we return
        ! the register SYMID built by GET_REGISTER_SYMID instead.

        IF .SCOPEPTR EQL 0
        THEN
          BEGIN

            ! Determine whether this name could be a register name.

            REGISTER_SYMID = GET_REGISTER_SYMID(.PATHNAME,
              .SCOPE_START_PTR, .REG_LINE_LEX_PTR);

            ! If this is not a register name, return the invalid symbol
            ! status to the caller. Note that we also give the informa-
            ! tional 'no line nn' message here if a line number was speci-
            ! fied which could not be found in the first scope.

            IF .REGISTER_SYMID EQL 0
            THEN
              BEGIN
                IF .HAVE_LINE_NUM AND (.FIRST_MODPTR NEQ 0)
                THEN
                  DBG$LINE_TO_PC_LOOKUP(.LINE_NUM, .STMT_NUM,
                    .FIRST_MODPTR, LINESTART, LINEEND, TRUE);

                SYMID[0] = 0;
                KIND[0] = RST$K_INVALID;
                RETURN;
                END;

            ! This symbol is a register. Return its SYMID and kind to the
            ! caller.

            SYMID[0] = .REGISTER_SYMID;
            KIND[0] = .REGISTER_SYMID[RST$B_KIND];
```

```
: 1550      1672  5      ! Zero the output parameters saying what scope we found the
: 1551      1673  5      ! symbol in - these are meaningless for registers.
: 1552      1674  5
: 1553      1675  5      IF .OUT_SCOPE_STATE NEQ 0
: 1554      1676  5      THEN
: 1555      1677  5          .OUT_SCOPE_STATE = 0;
: 1556      1678  5      IF .OUT_SCOPE NEQ 0
: 1557      1679  5      THEN
: 1558      1680  5          .OUT_SCOPE = 0;
: 1559      1681  5
: 1560      1682  5
: 1561      1683  5      ! Mark the register RST entry as being referenced by adding its
: 1562      1684  5      ! address to the RST Reference List (RST$REF_LIST). This says
: 1563      1685  5      ! that the RST entry is referenced by the current Debug command.
: 1564      1686  5      ! Then return.
: 1565      1687  5
: 1566      1688  5      IF .RST$REF_LIST[1] EQL .RST$REF_LIST[0]
: 1567      1689  5      THEN
: 1568      1690  6          BEGIN
: 1569      1691  6              RST$REF_LIST[0] = .RST$REF_LIST[0] + 20;
: 1570      1692  6              NEWREFLIST = DBG$GET_MEMORY(.RST$REF_LIST[0] + 2);
: 1571      1693  6              CH$MOVE(4*(.RST$REF_LIST[1] + 2), .RST$REF_LIST, .NEWREFLIST);
: 1572      1694  6              DBG$REL_MEMORY(.RST$REF_LIST);
: 1573      1695  6              RST$REF_LIST = .NEWREFLIST;
: 1574      1696  5          END;
: 1575      1697  5
: 1576      1698  5      RST$REF_LIST[1] = .RST$REF_LIST[1] + 1;
: 1577      1699  5      RST$REF_LIST[.RST$REF_LIST[1] + 1] = .RST$PTR;
: 1578      1700  5      RETURN;
: 1579      1701  4      END;
: 1580      1702  4
: 1581      1703  4
: 1582      1704  4      ! Set REG_SCOPE to TRUE if the current scope is the scope in which
: 1583      1705  4      ! a register would be looked up.
: 1584      1706  4
: 1585      1707  4      REG_SCOPE = .SCOPEPTR EQL .SCOPE_START_PTR;
: 1586      1708  4
: 1587      1709  4
: 1588      1710  4      ! Try to select a scope to search based on the current scope state.
: 1589      1711  4
: 1590      1712  4      SCOPE_STATE = .SCOPEPTR[SCOPE$L_STATE];
: 1591      1713  4      CASE .SCOPE_STATE FROM SCOPE$K_NORMAL TO SCOPE$K_SETMODS OF
: 1592      1714  4          SET
: 1593      1715  4
: 1594      1716  4
: 1595      1717  4      ! Search a normal, named scope as declared with a SET SCOPE
: 1596      1718  4      ! command. We pick up the scope information directly from the
: 1597      1719  4      ! scope list entry. Note that the scope's module must be SET;
: 1598      1720  4      ! otherwise the scope is not searched.
: 1599      1721  4
: 1600      1722  4      [SCOPE$K_NORMAL]:
: 1601      1723  5          BEGIN
: 1602      1724  5              SCOPE = .SCOPEPTR[SCOPE$L_RSTPTR];
: 1603      1725  5              MODRSTPTR = .SCOPEPTR[SCOPE$L_MODPTR];
: 1604      1726  5              IF .MODRSTPTR[RST$V_MODSET] THEN HAVE_SCOPE = TRUE;
: 1605      1727  5              SCOPEPTR = .SCOPEPTR[SCOPE$L_FLINK];
: 1606      1728  4          END;
```

```
1607 1729 4
1608 1730 4
1609 1731 4
1610 1732 4
1611 1733 4
1612 1734 4
1613 1735 4
1614 1736 4
1615 1737 4
1616 1738 5
1617 1739 5
1618 1740 5
1619 1741 5
1620 1742 5
1621 1743 4
1622 1744 4
1623 1745 4
1624 1746 4
1625 1747 4
1626 1748 4
1627 1749 4
1628 1750 4
1629 1751 4
1630 1752 5
1631 1753 5
1632 1754 5
1633 1755 6
1634 1756 6
1635 1757 5
1636 1758 6
1637 1759 6
1638 1760 6
1639 1761 6
1640 1762 6
1641 1763 7
1642 1764 7
1643 1765 7
1644 1766 7
1645 1767 7
1646 1768 7
1647 1769 7
1648 1770 7
1649 1771 7
1650 1772 7
1651 1773 7
1652 1774 7
1653 1775 7
1654 1776 7
1655 1777 7
1656 1778 6
1657 1779 6
1658 1780 5
1659 1781 5
1660 1782 5
1661 1783 4
1662 1784 4
1663 1785 4
```

```
! Search a 'numbered scope', i.e. the scope where the PC is cur-
! rently positioned N levels down in the CALL stack. To do this
! we look up the PC in the Static Address Table to find the con-
! taining lexical entity. If that succeeds (and the module is
! SET), we use that scope.
```

```
[SCOPE$K_NUMBERED]:
```

```
  BEGIN
    DBG$STA_NUMBERED_SCOPE(.SCOPEPTR[SCOPE$L_MODPTR],
                          MODRSTPTR, SCOPE, NUMSCP_INVOC_NUM);
    IF .SCOPE NEQ 0 THEN HAVE_SCOPE = TRUE;
    SCOPEPTR = .SCOPEPTR[SCOPE$L_FLINK];
  END;
```

```
! Search the Global Symbol Table (GST) for the symbol. We do
! this only if the symbol is of the form 'X' or '\X'. We do
! the search right here, and if we find the symbol, we return
! to the caller right away with the proper SYMID and KIND.
```

```
[SCOPE$K_GLOBAL]:
```

```
  BEGIN
    PNAME = .PATHVE[0];
    IF (.PATHNAME[PTH$B_TOTCNT] EQL .PATHNAME[PTH$B_PATHCNT])AND
      ((.PATHNAME[PTH$B_TOTCNT] EQL 2 AND .PNAME[0] EQL 0) OR
      (.PATHNAME[PTH$B_TOTCNT] EQL 1))
    THEN
      BEGIN
        RSTPTR = DBG$STA_LOOKUP_GBL(
                      .PATHVE[.PATHNAME[PTH$B_TOTCNT] - 1]);
        IF .RSTPTR NEQ 0
        THEN
          BEGIN
            SYMID[0] = .RSTPTR;
            KIND[0] = .RSTPTR[RST$B_KIND];

            ! If the user requested the information then fill in the
            ! output parameters which say what scope we are looking in.
            IF .OUT_SCOPE_STATE NEQ 0
            THEN
              .OUT_SCOPE_STATE = SCOPE$K_GLOBAL;
            IF .OUT_SCOPE NEQ 0
            THEN
              .OUT_SCOPE = 0;

            RETURN;
          END;
        END;
      END;
    SCOPEPTR = .SCOPEPTR[SCOPE$L_FLINK];
  END;
```

: 1664 1786 4
: 1665 1787 4
: 1666 1788 4
: 1667 1789 4
: 1668 1790 4
: 1669 1791 4
: 1670 1792 5
: 1671 1793 5
: 1672 1794 5
: 1673 1795 5
: 1674 1796 5
: 1675 1797 5
: 1676 1798 5
: 1677 1799 5
: 1678 1800 6
: 1679 1801 6
: 1680 1802 7
: 1681 1803 7
: 1682 1804 7
: 1683 1805 6
: 1684 1806 6
: 1685 1807 5
: 1686 1808 5
: 1687 1809 5
: 1688 1810 5
: 1689 1811 5
: 1690 1812 5
: 1691 1813 5
: 1692 1814 5
: 1693 1815 5
: 1694 1816 5
: 1695 1817 5
: 1696 1818 6
: 1697 1819 6
: 1698 1820 6
: 1699 1821 6
: 1700 1822 5
: 1701 1823 5
: 1702 1824 5
: 1703 1825 5
: 1704 1826 4
: 1705 1827 4
: 1706 1828 4
: 1707 1829 4
: 1708 1830 4
: 1709 1831 4
: 1710 1832 4
: 1711 1833 4
: 1712 1834 4
: 1713 1835 4
: 1714 1836 3
: 1715 1837 3
: 1716 1838 3
: 1717 1839 3
: 1718 1840 3
: 1719 1841 3
: 1720 1842 3

```
! Search all SET modules for the symbol. Here we locate the
! next SET module and use that as the current scope. Note that
! we accumulate candidate symbols over all SET modules before
! selecting the candidate that best matches the name.
[SCOPE$K SETMODS]:
  BEGIN

    ! The first time through, make NEXTSETMOD point to the first
    ! SET module and set the number of candidates to zero.
    IF .NEXTSETMOD EQL .RST$START_ADDR
    THEN
      BEGIN
        WHILE .NEXTSETMOD NEQ 0 DO
          BEGIN
            IF .NEXTSETMOD[RST$V MODSET] THEN EXITLOOP;
            NEXTSETMOD = .NEXTSETMOD[RST$L_NXTMODPTR];
          END;
        END;

        ! Make MODRSTPTR and SCOPE point to the next SET module and
        ! make NEXTSETMOD point to the SET module we will search the
        ! next time around. When NEXTSETMOD becomes zero, there is
        ! no next time around.
        MODRSTPTR = .NEXTSETMOD;
        SCOPE = .MODRSTPTR;
        WHILE .NEXTSETMOD NEQ 0 DO
          BEGIN
            NEXTSETMOD = .NEXTSETMOD[RST$L_NXTMODPTR];
            IF .NEXTSETMOD EQL 0 THEN EXITLOOP;
            IF .NEXTSETMOD[RST$V_MODSET] THEN EXITLOOP;
          END;

          IF .MODRSTPTR NEQ 0 THEN HAVE_SCOPE = TRUE;
          IF .NEXTSETMOD EQL 0 THEN SCOPEPTR = .SCOPEPTR[SCOPE$L_FLINK];
        END;

      TES;

      ! If we now have a scope to search, exit the scope-locating loop
      ! and search that scope. Otherwise, loop to locate another scope.
      IF .HAVE_SCOPE THEN EXITLOOP;

    END;

    ! End of WHILE loop to find a scope

    ! We now have a scope to search. Make sure the corresponding module's
    ! symbol table is in the RST.
    IF NOT .MODRSTPTR[RST$V_MOD_IN_RST]
```

```
1721 1843 3
1722 1844 3
1723 1845 3
1724 1846 3
1725 1847 3
1726 1848 3
1727 1849 3
1728 1850 3
1729 1851 3
1730 1852 3
1731 1853 3
1732 1854 3
1733 1855 3
1734 1856 3
1735 1857 3
1736 1858 3
1737 1859 3
1738 1860 3
1739 1861 3
1740 1862 3
1741 1863 3
1742 1864 4
1743 1865 4
1744 1866 4
1745 1867 4
1746 1868 4
1747 1869 4
1748 1870 4
1749 1871 4
1750 1872 4
1751 1873 4
1752 1874 4
1753 1875 4
1754 1876 4
1755 1877 4
1756 1878 4
1757 1879 4
1758 1880 4
1759 1881 4
1760 1882 4
1761 1883 4
1762 1884 4
1763 1885 4
1764 1886 4
1765 1887 4
1766 1888 5
1767 1889 5
1768 1890 5
1769 1891 5
1770 1892 6
1771 1893 6
1772 1894 5
1773 1895 6
1774 1896 6
1775 1897 6
1776 1898 6
1777 1899 6
```

```
THEN
  DBG$RST_BUILD(.MODRSTPTR, FALSE);

! If the user requested the information then fill in the
! output parameters which say what scope we are looking in.
IF .OUT_SCOPE_STATE NEQ 0
THEN
  .OUT_SCOPE_STATE = .SCOPE_STATE;
IF .OUT_SCOPE NEQ 0
THEN
  .OUT_SCOPE = .SCOPE;

! If there is a line number in the pathname, find the lexical entity
! within this scope's module which contains that line number. Note
! that we search for the lowest level (innermost) lexical entity.
IF .HAVE_LINE_NUM
THEN
  BEGIN

    ! If this is the first real scope on the scope list, save the
    ! current Module RST Entry pointer in case we will need it for
    ! the 'no line nnn' informational message.
    IF .FIRST_MODPTR EQL 0 THEN FIRST_MODPTR = .MODRSTPTR;

    ! Look up the line and statement numbers in the scope's module.
    STATUS = DBG$LINE_TO_PC_LOOKUP(.LINE_NUM, .STMT_NUM,
    .MODRSTPTR, LINESTART, LINEEND, FALSE);

    ! Look up the lowest-level (innermost) lexical entity which contains
    ! the line we just looked up. We do this by searching the module's
    ! Static Address Table.
    SATPTR = .MODRSTPTR[RST$L SAT PTR];
    IF NOT .STATUS THEN SATPTR = 0;
    LINE_LEX_PTR = 0;
    WHILE .SATPTR NEQ 0 DO
      BEGIN
        IF .SATPTR[SAT$L START] GTR .LINESTART THEN EXITLOOP;
        RSTPTR = .SATPTR[SAT$L RSTPTR];
        IF (.SATPTR[SAT$L_END] GEQ .LINESTART) AND
        (.RSTPTR[RST$B_KIND] EQL RST$K_ROUTINE OR
        .RSTPTR[RST$B_KIND] EQL RST$K_BLOCK)
        THEN
          BEGIN
            IF .LINE_LEX_PTR EQL 0
            THEN
              LINE_LEX_PTR = .RSTPTR
```



```

: 1778      1900 6
: 1779      1901 7
: 1780      1902 7
: 1781      1903 7
: 1782      1904 8
: 1783      1905 8
: 1784      1906 8
: 1785      1907 9
: 1786      1908 9
: 1787      1909 9
: 1788      1910 8
: 1789      1911 8
: 1790      1912 8
: 1791      1913 7
: 1792      1914 7
: 1793      1915 6
: 1794      1916 6
: 1795      1917 5
: 1796      1918 5
: 1797      1919 5
: 1798      1920 5
: 1799      1921 4
: 1800      1922 4
: 1801      1923 4
: 1802      1924 4
: 1803      1925 4
: 1804      1926 4
: 1805      1927 4
: 1806      1928 4
: 1807      1929 3
: 1808      1930 3
: 1809      1931 3
: 1810      1932 3
: 1811      1933 3
: 1812      1934 3
: 1813      1935 3
: 1814      1936 3
: 1815      1937 3
: 1816      1938 3
: 1817      1939 4
: 1818      1940 4
: 1819      1941 4
: 1820      1942 4
: 1821      1943 4
: 1822      1944 4
: 1823      1945 4
: 1824      1946 4
: 1825      1947 4
: 1826      1948 5
: 1827      1949 5
: 1828      1950 5
: 1829      1951 5
: 1830      1952 5
: 1831      1953 5
: 1832      1954 5
: 1833      1955 5
: 1834      1956 5

ELSE
  BEGIN
    RPTR = .RSTPTR;
    WHILE .RPTR[RST$B_KIND] NEQ RST$K_MODULE DO
      BEGIN
        IF .RPTR EQL .LINE_LEX_PTR
        THEN
          BEGIN
            LINE_LEX_PTR = .RSTPTR;
            EXIT[COOP];
          END;
        RPTR = .RPTR[RST$L_UPSCOPEPTR];
      END;
    END;
  END;

  SATPTR = .SATPTR[SAT$L_FLINK];
END;
! End of WHILE loop over the SAT

! In case we have to look up a register in this scope, save the
! value of the line number lexical entity pointer.
IF .REG_SCOPE THEN REG_LINE_LEX_PTR = .LINE_LEX_PTR;
END;
! End of line number lexical entity code

! Set up the RST Hash Table search for this symbol and loop over all
! hash table entries for the symbol's name. For each RST entry we find,
! we try to match the full pathname. If this succeeds and the symbol is
! in the current scope, the RST entry is added to a "candidate list".
DBG$HASH_FIND_SETUP(.NAMEPTR);
WHILE TRUE DO
  BEGIN
    ! Get the next RST entry with the specified symbol name. If the
    ! desired symbol is a line number, we pick up the lexical entity
    ! which contains the line instead.
    IF .LINE_NUM_IS_LAST
    THEN
      BEGIN
        RSTPTR = .LINE_LEX_PTR;
        LINE_LEX_PTR = 0;
      END
    ! Otherwise, pick up the next RST Hash Table entry with the
    ! specified symbol name.
```

```
1835 1957 4
1836 1958 4
1837 1959 4
1838 1960 4
1839 1961 4
1840 1962 4
1841 1963 4
1842 1964 4
1843 1965 4
1844 1966 4
1845 1967 4
1846 1968 4
1847 1969 4
1848 1970 4
1849 1971 4
1850 1972 4
1851 1973 4
1852 1974 4
1853 1975 5
1854 1976 5
1855 1977 5
1856 1978 5
1857 1979 5
1858 1980 5
1859 1981 5
1860 1982 5
1861 1983 6
1862 1984 5
1863 1985 5
1864 1986 5
1865 1987 5
1866 1988 5
1867 1989 5
1868 1990 5
1869 1991 5
1870 1992 5
1871 1993 6
1872 1994 6
1873 1995 7
1874 1996 6
1875 1997 6
1876 1998 6
1877 1999 5
1878 2000 5
1879 2001 5
1880 2002 5
1881 2003 5
1882 2004 5
1883 2005 5
1884 2006 5
1885 2007 5
1886 2008 5
1887 2009 5
1888 2010 5
1889 2011 5
1890 2012 5
1891 2013 5
```

```
ELSE
    RSTPTR = DBG$HASH_FIND(.NAMEPTR);

    ! If the RST pointer is zero, we found no more symbols with the
    ! right name so we exit the search loop for this scope.
    IF .RSTPTR EQL 0 THEN EXITLOOP;

    ! Loop through the RST entry's scope chain to match it to the speci-
    ! fied pathname. If the full pathname matches and the symbol is in
    ! the current scope, we add the RST entry to the "candidate list".
    STKPTR = 0;
    RPTR = .RSTPTR;
    PINDEX = .PATHNAME[PTHSB_TOTCNT];
    WHILE TRUE DO
        BEGIN

            ! If this is a global symbol or a module, do not even attempt to
            ! match it to the pathname--exit the pathname matching loop now.
            IF .RSTPTR[RST$V_GLOBAL] OR
                (.RSTPTR[RST$B_KIND] EQL RST$K_MODULE) OR
                ((NOT .TYPE_FLAG) AND (.RSTPTR[RST$B_KIND] EQL RST$K_TYPE))
            THEN
                EXITLOOP;

            ! Also, if we are called by DBG$RST_SETSCOPE, do not consider
            ! the symbol unless it is a routine or lexical block.
            IF .SET_SCOPE
            THEN
                BEGIN
                    IF (.RSTPTR[RST$B_KIND] NEQ RST$K_ROUTINE) AND
                        (.RSTPTR[RST$B_KIND] NEQ RST$K_BLOCK)
                    THEN
                        EXITLOOP;
                END;

            ! Make a new SYMSTACK entry for this RST entry in the up-scope
            ! chain.
            STKPTR = .STKPTR + 1;
            IF .STKPTR GEQ MAX_STACK THEN EXITLOOP;
            SYMSTACK[.STKPTR, STK_RSTPTR] = .RPTR;
            SYMSTACK[.STKPTR, STK_PINDEX] = 0;
            SYMSTACK[.STKPTR, STK_IPINDEX] = 0;

            ! If this pathname component is a line number or a scope number,
            ! we skip over it in pathname matching.
```

1892 2014 5
1893 2015 5
1894 2016 6
1895 2017 5
1896 2018 5
1897 2019 5
1898 2020 5
1899 2021 5
1900 2022 5
1901 2023 5
1902 2024 5
1903 2025 5
1904 2026 5
1905 2027 5
1906 2028 5
1907 2029 5
1908 2030 6
1909 2031 5
1910 2032 6
1911 2033 6
1912 2034 6
1913 2035 6
1914 2036 6
1915 2037 6
1916 2038 6
1917 2039 6
1918 2040 6
1919 2041 6
1920 2042 6
1921 2043 6
1922 2044 6
1923 2045 7
1924 2046 6
1925 2047 7
1926 2048 7
1927 2049 7
1928 2050 7
1929 2051 7
1930 2052 7
1931 2053 7
1932 2054 7
1933 2055 7
1934 2056 7
1935 2057 7
1936 2058 7
1937 2059 7
1938 2060 7
1939 2061 7
1940 2062 7
1941 2063 7
1942 2064 7
1943 2065 7
1944 2066 7
1945 2067 7
1946 2068 7
1947 2069 7
1948 2070 8

```
!
IF (.HAVE_LINE_NUM AND (.PINDEX EQL .LINE_NUM_LOC)) OR
(.HAVE_NUM_SCOPE AND (.PINDEX EQL 1))
THEN
    PINDEX = .PINDEX - 1;

! If the current pathname component matches the current scope
! chain name, set PINDEX to point to the next pathname compo-
! nent. If PINDEX already pointed to the top component name,
! the pathname matches and we make a candidate list entry.
PNAME = .PATHVEC[PINDEX - 1];
IF .PINDEX EQL 0 THEN PNAME = .PATHVEC[0];
RNAME = DBG$GET_DST_NAME(.RPTR[RST$L_DSTPTR]);
IF CH$EQL(PNAME[0], PNAME[1], .RNAME[0], RNAME[1], 0) OR
(.PINDEX EQL 0)
THEN
    BEGIN

        ! Record the fact that RPTR matches this Pathname component.
        SYMSTACK[.STKPTR, STK_PINDEX] = .PINDEX;

        ! If the last (top-level) pathname component just matched,
        ! we see if the symbol is in the current scope. If it is,
        ! we add the symbol to the candidate list (CANDLIST).
        IF (.PINDEX LEQ .PATHNAME[PTH$B_PATHCNT]) AND
        (.RPTR[RST$B_KIND] NEQ RST$K_TYPCOMP)
        THEN
            BEGIN

                ! Determine what the scope of the current symbol is.
                SYMSCOPE = .RSTPTR;
                IF .RSTPTR[RST$B_KIND] NEQ RST$K_MODULE
                THEN
                    SYMSCOPE = .RSTPTR[RST$L_UPSCOPEPTR];

                IF .SYMSCOPE[RST$B_KIND] EQL RST$K_TYPE
                THEN
                    SYMSCOPE = .SYMSCOPE[RST$L_UPSCOPEPTR];

                ! If we are searching all set modules, we claim that the
                ! the symbol is declared at the module level so that all
                ! symbols have the same definition depth. Also, if we
                ! are looking for a line number, we treat it as being
                ! defined at the module level.
                IF .SCOPE_STATE EQL SCOPE$K_SETMODS OR .LINE_NUM_IS_LAST
                THEN
                    BEGIN
```

1949	2071	8
1950	2072	8
1951	2073	8
1952	2074	7
1953	2075	7
1954	2076	7
1955	2077	7
1956	2078	7
1957	2079	7
1958	2080	7
1959	2081	7
1960	2082	7
1961	2083	8
1962	2084	8
1963	2085	8
1964	2086	8
1965	2087	9
1966	2088	9
1967	2089	9
1968	2090	8
1969	2091	8
1970	2092	8
1971	2093	8
1972	2094	7
1973	2095	7
1974	2096	7
1975	2097	7
1976	2098	7
1977	2099	7
1978	2100	7
1979	2101	7
1980	2102	7
1981	2103	8
1982	2104	8
1983	2105	8
1984	2106	8
1985	2107	9
1986	2108	9
1987	2109	9
1988	2110	10
1989	2111	10
1990	2112	10
1991	2113	9
1992	2114	9
1993	2115	9
1994	2116	8
1995	2117	8
1996	2118	7
1997	2119	7
1998	2120	7
1999	2121	7
2000	2122	7
2001	2123	7
2002	2124	7
2003	2125	7
2004	2126	7
2005	2127	8

```
WHILE .SYMSCOPE[RST$B_KIND] NEQ RST$K_MODULE DO
    SYMSCOPE = .SYMSCOPE[RST$L_UPSCOPEPTR];

END;

! Determine whether the symbol is in the current scope.
!
SCPTR = .SCOPE;
DEFDEPTH = 0;
IN_SCOPE = TRUE;
WHILE TRUE DO
    BEGIN
        IF .SCPTR EQL .SYMSCOPE THEN EXITLOOP;
        IF .SCPTR[RST$B_KIND] EQL RST$K_MODULE
            THEN
                BEGIN
                    IN_SCOPE = FALSE;
                    EXITLOOP;
                END;
        SCPTR = .SCPTR[RST$L_UPSCOPEPTR];
        DEFDEPTH = .DEFDEPTH + 1;
    END;

! If a line number is present in the pathname, make sure
! this symbol has the line's lexical entity in its up-
! scope chain. Otherwise set IN_SCOPE to FALSE.
!
IF .HAVE_LINE_NUM AND .IN_SCOPE AND NOT .LINE_NUM_IS_LAST
THEN
    BEGIN
        IN_SCOPE = FALSE;
        SCPTR = .RSTPTR;
        WHILE .SCPTR[RST$B_KIND] NEQ RST$K_MODULE DO
            BEGIN
                IF .SCPTR EQL .LINE_LEX_PTR
                THEN
                    BEGIN
                        IN_SCOPE = TRUE;
                        EXITLOOP;
                    END;
            SCPTR = .SCPTR[RST$L_UPSCOPEPTR];
        END;
    END;

! If the symbol is in the current scope, create a "can-
! didate entry" for it. Then enter that entry on the
! "candidate list".
!
IF .IN_SCOPE
THEN
    BEGIN
```

```
: 2006      2128      8
: 2007      2129      8
: 2008      2130      8
: 2009      2131      8
: 2010      2132      8
: 2011      2133      8
: 2012      2134      8
: 2013      2135      9
: 2014      2136      9
: 2015      2137      9
: 2016      2138     10
: 2017      2139     10
: 2018      2140     10
: 2019      2141     10
: 2020      2142      9
: 2021      2143      9
: 2022      2144      8
: 2023      2145      8
: 2024      2146      8
: 2025      2147      8
: 2026      2148      8
: 2027      2149      8
: 2028      2150      8
: 2029      2151      8
: 2030      2152      8
: 2031      2153      8
: 2032      2154      8
: 2033      2155      8
: 2034      2156      8
: 2035      2157      9
: 2036      2158      9
: 2037      2159      9
: 2038      2160      9
: 2039      2161      9
: 2040      2162      9
: 2041      2163      8
: 2042      2164      8
: 2043      2165      8
: 2044      2166      7
: 2045      2167      7
: 2046      2168      7
: 2047      2169      7
: 2048      2170      7
: 2049      2171      7
: 2050      2172      7
: 2051      2173      7
: 2052      2174      7
: 2053      2175      7
: 2054      2176      8
: 2055      2177      8
: 2056      2178      8
: 2057      2179      9
: 2058      2180      9
: 2059      2181      9
: 2060      2182      9
: 2061      2183      8
: 2062      2184      8
```

```
: Create the candidate entry for the symbol.
CANDBLK = DBG$GET_MEMORY(CAND_ENTSIZ*(.STKPTR+1));
J = 0;
INCR I FROM 1 TO .STKPTR DO
  BEGIN
    IF .SYMSTACK[I, STK_TPINDEX] EQL 0
    THEN
      BEGIN
        CANDBLK[J, CAND_RSTPTR] = .SYMSTACK[I, STK_RSTPTR];
        CANDBLK[J, CAND_PINDEX] = .SYMSTACK[I, STK_PINDEX];
        J = J + 1;
      END;
    END;
CANDBLK[J, CAND_RSTPTR] = 0;
CANDBLK[J, CAND_PINDEX] = .DEFDEPTH;

: Enter the candidate entry on the candidate list.
: Note that we expand the candidate list memory
: block if it is too small.
NCANDS = .NCANDS + 1;
IF .NCANDS GTR .CANDLST[0]
THEN
  BEGIN
    CANDLST[0] = .CANDLST[0] + 10;
    OLDCAND = .CANDLST;
    CANDLST = DBG$GET_MEMORY(.CANDLST[0] + 1);
    CH$MOVE(4*.NCANDS, .OLDCAND, .CANDLST);
    DBG$REL_MEMORY(.OLDCAND);
  END;
CANDLST[.NCANDS] = .CANDBLK;
END;

: Now tear down SYMSTACK until we get to the bottom or
: until we get to a TYPE entry whose type reference
: table has not been exhausted. If no such entry is
: found, we exit the pathname match loop (with STKPTR =
: 0) for this hash table symbol.
WHILE .STKPTR GTR 0 DO
  BEGIN
    IF .SYMSTACK[.STKPTR, STK_TPINDEX] NEQ 0
    THEN
      BEGIN
        TPINDEX = .SYMSTACK[.STKPTR, STK_TPINDEX];
        RPTR = .SYMSTACK[.STKPTR, STK_RSTPTR];
        IF .TPINDEX LSS .RPTR[RS$W_TYPREFCNT] THEN EXITLOOP;
      END;
```

```

: 2063      2185      8
: 2064      2186      7
: 2065      2187      7
: 2066      2188      7
: 2067      2189      7
: 2068      2190      7
: 2069      2191      7
: 2070      2192      7
: 2071      2193      7
: 2072      2194      7
: 2073      2195      7
: 2074      2196      7
: 2075      2197      7
: 2076      2198      7
: 2077      2199      7
: 2078      2200      7
: 2079      2201      6
: 2080      2202      6
: 2081      2203      6
: 2082      2204      6
: 2083      2205      6
: 2084      2206      6
: 2085      2207      6
: 2086      2208      6
: 2087      2209      6
: 2088      2210      5
: 2089      2211      5
: 2090      2212      5
: 2091      2213      5
: 2092      2214      5
: 2093      2215      5
: 2094      2216      5
: 2095      2217      5
: 2096      2218      5
: 2097      2219      5
: 2098      2220      5
: 2099      2221      5
: 2100      2222      5
: 2101      2223      5
: 2102      2224      5
: 2103      2225      5
: 2104      2226      5
: 2105      2227      5
: 2106      2228      6
: 2107      2229      6
: 2108      2230      6
: 2109      2231      6
: 2110      2232      6
: 2111      2233      6
: 2112      2234      6
: 2113      2235      6
: 2114      2236      6
: 2115      2237      5
: 2116      2238      5
: 2117      2239      4
: 2118      2240      4
: 2119      2241      3

      STKPTR = .STKPTR - 1;
      END;

      IF .STKPTR EQL 0 THEN EXITLOOP;

      ! If we found a type entry in SYMSTACK whose reference
      ! table is not exhausted, we reset RPTR and PINDEX to
      ! continue generating possible candidates from symbols
      ! of this type.
      SYMSTACK[.STKPTR, STK_TPINDEX] = .TPINDEX + 1;
      TPTR = .RPTR[RST$L_TYPREFTBL];
      RPTR = .TPTR[.TPINDEX];
      PINDEX = .SYMSTACK[.STKPTR, STK_PINDEX] + 1;

      END;          ! End of .PINDEX LEQ ... THEN clause

      ! We have more pathname components to match. Decrement
      ! PINDEX and stay in the pathname matching loop.
      PINDEX = .PINDEX - 1;
      IF .PINDEX LSS 0 THEN PINDEX = 0;

      END;          ! End of CH$EQL test's THEN clause

      ! If the RST entry's up-scope chain has ended, we exit the path-
      ! name matching loop. Otherwise, we link up the up-scope chain
      ! and continue pathname matching.
      IF .RPTR[RST$B_KIND] EQL RST$K_MODULE THEN EXITLOOP;
      RPTR = .RPTR[RST$L_UPSCOPEPTR];

      ! If the up-scope symbol is a Type RST Entry, we set up an entry
      ! for it on SYMSTACK. This stack entry will enable us to try
      ! all possible symbols of this type as the up-scope continuation
      ! of a type component (e.g., record or variant component).
      IF .RPTR[RST$B_KIND] EQL RST$K_TYPE
      THEN
      BEGIN
      IF .RPTR[RST$L_TYPREFTBL] EQL 0 THEN EXITLOOP;
      STKPTR = .STKPTR + 1;
      IF .STKPTR GEQ MAX_STACK THEN EXITLOOP;
      SYMSTACK[.STKPTR, STK_RSTPTR] = .RPTR;
      SYMSTACK[.STKPTR, STK_PINDEX] = .PINDEX;
      SYMSTACK[.STKPTR, STK_TPINDEX] = 1;
      TPTR = .RPTR[RST$L_TYPREFTBL];
      RPTR = .TPTR[0];
      END;

      END;          ! End of pathname matching WHILE loop

      END;          ! End of WHILE loop over hash table
```

```
2120 2242 3
2121 2243 3
2122 2244 3
2123 2245 3
2124 2246 3
2125 2247 3
2126 2248 3
2127 2249 3
2128 2250 3
2129 2251 3
2130 2252 4
2131 2253 3
2132 2254 4
2133 2255 4
2134 2256 4
2135 2257 4
2136 2258 4
2137 2259 4
2138 2260 4
2139 2261 4
2140 2262 4
2141 2263 4
2142 2264 4
2143 2265 4
2144 2266 4
2145 2267 4
2146 2268 4
2147 2269 4
2148 2270 4
2149 2271 4
2150 2272 4
2151 2273 4
2152 2274 5
2153 2275 5
2154 2276 5
2155 2277 6
2156 2278 5
2157 2279 5
2158 2280 5
2159 2281 5
2160 2282 5
2161 2283 4
2162 2284 4
2163 2285 4
2164 2286 4
2165 2287 4
2166 2288 4
2167 2289 4
2168 2290 4
2169 2291 4
2170 2292 4
2171 2293 4
2172 2294 4
2173 2295 4
2174 2296 4
2175 2297 4
2176 2298 4
```

```
! We now have a list of candidate symbols which are in the current scope
! and which may match the pathname. Unless the list is empty, call a
! language-specific routine to select the candidate symbol which best
! matches the pathname. Note that when we search all SET modules, we
! do not call this selection routine until candidate symbols have been
! accumulated from all SET modules.
```

```
IF (.NCANDS GTR 0) AND
   (.SCOPE_STATE NEQ SCOPE$K_SETMODS OR .NEXTSETMOD EQL 0)
THEN
```

```
  BEGIN
```

```
    CASE .DBG$GB_LANGUAGE FROM DBG$K_MACRO TO DBG$K_UNKNOWN OF
      SET
```

```
      ! Handle languages with "normal" scope rules--data qualification
      ! must be complete, or it is absent from the language.
```

```
      [DBG$K_MACRO, DBG$K_FORTRAN,
       DBG$K_BLISS, DBG$K_BASIC,
       DBG$K_PASCAL, DBG$K_C,
       INRANGE, OUTRANGE]:
        GOOD_CAND = SCOPE_RULE_NORMAL(.PATHNAME, .NCANDS,
                                       .CANDLIST, .ARRAY_FLAG);
```

```
      ! Handle COBOL scope rules--data qualification need not be
      ! complete and is resolved by COBOL scope rules.
```

```
      [DBG$K_COBOL]:
        BEGIN
          SCPTR = 0;
          IF (.SCOPE_STATE EQL SCOPE$K_NORMAL) OR
             (.SCOPE_STATE EQL SCOPE$K_NUMBERED)
          THEN
            SCPTR = .SCOPE;

            GOOD_CAND = SCOPE_RULE_COBOL(.PATHNAME,
                                         .NCANDS, .CANDLIST, .SCPTR);
          END;
```

```
      ! Handle PL/I scope rules--data qualification need not be
      ! complete and is resolved by PL/I rules.
```

```
      [DBG$K_PLI]:
        GOOD_CAND = SCOPE_RULE_PLI(.PATHNAME, .NCANDS, .CANDLIST);
```

```
    TES;
```

```
! If we found a valid and unique match for the pathname in this
! scope, make CANDBLK point to that symbol and exit the scope
! search loop.
```

```
2177 2299 4      IF .GOOD_CAND GTR 0
2178 2300 4      THEN
2179 2301 5          BEGIN
2180 2302 5              CANDBLK = .CANDLST[.GOOD_CAND];
2181 2303 5              EXITLOOP;
2182 2304 4              END;
2183 2305 4
2184 2306 4
2185 2307 4      ! We did not find a valid and unique symbol. Release all candidate
2186 2308 4      ! blocks on the candidate list to the free memory pool.
2187 2309 4
2188 2310 4      INCR I FROM 1 TO .NCANDS DO DBG$REL_MEMORY(.CANDLST[I]);
2189 2311 4      NCANDS = 0;
2190 2312 4
2191 2313 4
2192 2314 4      ! If the symbol turns out not be unique, return the not-unique
2193 2315 4      ! code to KIND and a zero to SYMID; then return to the caller.
2194 2316 4      ! Otherwise, loop to locate another scope to search.
2195 2317 4
2196 2318 4      IF .GOOD_CAND EQL -1
2197 2319 4      THEN
2198 2320 5          BEGIN
2199 2321 5              SYMID[0] = 0;
2200 2322 5              KIND[0] = RST$K_NOTUNIQUE;
2201 2323 5              RETURN;
2202 2324 4              END;
2203 2325 4
2204 2326 3      END;
2205 2327 3
2206 2328 2      END;                                ! End of WHILE loop over all scopes
2207 2329 2
2208 2330 2
2209 2331 2      ! Now go through the symbol's candidate entry to create new Data RST Entries
2210 2332 2      ! from any Type Component RST Entries. These new RST entries represent this
2211 2333 2      ! specific instance of data qualification. These RST entries are put on the
2212 2334 2      ! Temporary RST Entry List.
2213 2335 2      !
2214 2336 2      J = 0;
2215 2337 2      WHILE TRUE DO
2216 2338 3          BEGIN
2217 2339 3              RPTR = .CANDBLK[J, CAND_RSTPTR];
2218 2340 3              IF .RPTR EQL 0 THEN EXITLOOP;
2219 2341 3              IF .RPTR[RST$B_KIND] NEQ RST$K_TYPCOMP THEN EXITLOOP;
2220 2342 3              RSTPTR = DBG$GET_MEMORY(RST$K_DATENTSIZ);
2221 2343 3              RSTPTR[RST$L_HASH_FLINK] = .RST$TEMP_LIST;
2222 2344 3              RST$TEMP_LIST = .RSTPTR;
2223 2345 3              RSTPTR[RST$L_DSTPTR] = .RPTR[RST$L_DSTPTR];
2224 2346 3              RSTPTR[RST$L_UPSCOPEPTR] = .RPTR[RST$L_UPSCOPEPTR];
2225 2347 3              RSTPTR[RST$B_KIND] = RST$K_INVALID;
2226 2348 3              RSTPTR[RST$L_TYPEPTR] = .RPTR[RST$L_TYPEPTR];
2227 2349 3              CANDBLK[J, CAND_RSTPTR] = .RSTPTR;
2228 2350 3              J = J + 1;
2229 2351 2          END;
2230 2352 2
2231 2353 2
2232 2354 2      ! Then make a second scan over the new Data RST Entries to fix up their
2233 2355 2      ! up-scope pointers.
```



```
2234 2356 2
2235 2357 2
2236 2358 2
2237 2359 2
2238 2360 3
2239 2361 3
2240 2362 3
2241 2363 3
2242 2364 3
2243 2365 3
2244 2366 3
2245 2367 3
2246 2368 3
2247 2369 2
2248 2370 2
2249 2371 2
2250 2372 2
2251 2373 2
2252 2374 2
2253 2375 2
2254 2376 2
2255 2377 3
2256 2378 3
2257 2379 3
2258 2380 3
2259 2381 3
2260 2382 3
2261 2383 3
2262 2384 3
2263 2385 3
2264 2386 2
2265 2387 2
2266 2388 2
2267 2389 2
2268 2390 2
2269 2391 2
2270 2392 2
2271 2393 2
2272 2394 2
2273 2395 2
2274 2396 2
2275 2397 2
2276 2398 3
2277 2399 3
2278 2400 3
2279 2401 3
2280 2402 3
2281 2403 3
2282 2404 3
2283 2405 3
2284 2406 3
2285 2407 3
2286 2408 4
2287 2409 4
2288 2410 4
2289 2411 5
2290 2412 5

J = 0;
WHILE TRUE DO
  BEGIN
    RSTPTR = .CANDBLK[J, CAND_RSTPTR];
    IF .RSTPTR EQL 0 THEN EXIT[OOP];
    IF .RSTPTR[RST$B_KIND] NEQ RST$K_INVALID THEN EXITLOOP;
    RSTPTR[RST$B_KIND] = RST$K_DATA;
    IF .CANDBLK[J + 1, CAND_RSTPTR] NEQ 0
    THEN
      RSTPTR[RST$L_UPSCOPEPTR] = .CANDBLK[J + 1, CAND_RSTPTR];

    J = J + 1;
  END;

! If the symbol is a line number, create the Line Number RST Entry for the
! symbol and make its address the symbol's SYMID.
!
IF .LINE_NUM_IS_LAST
THEN
  BEGIN
    MODRSTPTR = .CANDBLK[0, CAND_RSTPTR];
    WHILE .MODRSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
      MODRSTPTR = .MODRSTPTR[RST$L_UPSCOPEPTR];

    STATUS = DBG$LINE_TO_PC_LOOKUP(.LINE_NUM, .STMT_NUM,
      .MODRSTPTR, LINESTART, LINEEND, FALSE);
    CANDBLK[0, CAND_RSTPTR] = DBG$STA_LINE_NUM_RST(.CANDBLK[0, CAND_RSTPTR],
      .LINE_NUM, .STMT_NUM, .LINESTART, .LINEEND);
  END;

! Pick up the SYMID (RST pointer) of the symbol we found.
!
RSTPTR = .CANDBLK[0, CAND_RSTPTR];

! If there is an invocation number, check that the invocation number was
! applied to the inner-most routine in the up-scope chain. If that looks
! good, create an Invocation Number RST Entry for the symbol.
!
IF (.PATHNAME[PTH$B_LOCINVO] NEQ 0) AND (NOT .HAVE_NUM_SCOPE)
THEN
  BEGIN
    ! Find the inner-most routine containing the declaration of this symbol.
    ! This is the routine to which the invocation number must apply.
    !
    ROUTPTR = .RSTPTR;
    WHILE .ROUTPTR[RST$B_KIND] NEQ RST$K_ROUTINE DO
      BEGIN
        IF .ROUTPTR[RST$B_KIND] EQL RST$K_MODULE
        THEN
          BEGIN
            DBG$NPATHDESC_TO_CS(.PATHNAME, PATHSTRING);
```

```
2291 2413 5          SIGNAL(DBG$_MISINVNUM, 1, .PATHSTRING);
2292 2414 4          END;
2293 2415 4
2294 2416 4          ROUTPTR = .ROUTPTR[RST$$_UPSCOPEPTR];
2295 2417 3          END;
2296 2418 3
2297 2419 3
2298 2420 3          ! Now make sure the invocation number was indeed appended to that
2299 2421 3          ! routine name in the pathname.
2300 2422 3
2301 2423 3          PNAME = .PATHVEC[.PATHNAME[PTH$$_LOCINVO] - 1];
2302 2424 3          RNAME = DBG$$_GET_DST_NAME(.ROUTPTR[RST$$_DSTPTR]);
2303 2425 3          IF CH$$_NEQ(.PNAME[0], PNAME[1], .RNAME[0], RNAME[1], 0)
2304 2426 3          THEN
2305 2427 4              BEGIN
2306 2428 4                  DBG$$_NPATHDESC TO CS(.PATHNAME, PATHSTRING);
2307 2429 4                  SIGNAL(DBG$$_MISINVNUM, 1, .PATHSTRING);
2308 2430 3              END;
2309 2431 3
2310 2432 3
2311 2433 3          ! All looks good. Create the Invocation Number RST Entry along with a
2312 2434 3          ! new copy of the symbol's RST entry if the number is non-zero.
2313 2435 3
2314 2436 3          IF .PATHNAME[PTH$$_INVO] NEQ 0
2315 2437 3          THEN
2316 2438 3              RSTPTR = DBG$$_BUILD_INVOC_RST(.RSTPTR, .PATHNAME[PTH$$_INVO]);
2317 2439 3
2318 2440 3          END
2319 2441 3
2320 2442 3
2321 2443 3          ! If this symbol was specified with a numbered scope (i.e. 2\X) and the
2322 2444 3          ! invocation number is non-zero, create an Invocation Number RST Entry
2323 2445 3          ! for the symbol.
2324 2446 3
2325 2447 3          ELSE IF .HAVE_NUM_SCOPE AND (.NUMSCP_INVOC_NUM NEQ 0)
2326 2448 2          THEN
2327 2449 2              RSTPTR = DBG$$_BUILD_INVOC_RST(.RSTPTR, .NUMSCP_INVOC_NUM)
2328 2450 2
2329 2451 2
2330 2452 2          ! And also, if the symbol was a simple symbol without any pathname qualifi-
2331 2453 2          ! cation, do the proper up-level addressing (if any) in the scope we found
2332 2454 2          ! it in to get the proper invocation number for the symbol.
2333 2455 2
2334 2456 2          ELSE IF (.PATHNAME[PTH$$_LOCINVO] EQL 0) AND
2335 2457 2              (.PATHNAME[PTH$$_PATHCNT] EQL 1)
2336 2458 2          THEN
2337 2459 2              RSTPTR = FOLLOW_STATIC_LINK(.RSTPTR, .SCOPE);
2338 2460 2
2339 2461 2
2340 2462 2          ! Now return the selected symbol's SYMID and KIND to the caller.
2341 2463 2
2342 2464 2          SYMID[0] = .RSTPTR;
2343 2465 2          KIND[0] = .RSTPTR[RST$$_KIND];
2344 2466 2
2345 2467 2
2346 2468 2          ! Release all candidate blocks on the candidate list to the memory pool.
2347 2469 2
```

```
: 2348      2470 2      INCR I FROM 1 TO .NCANDS DO DBG$REL_MEMORY(.CANDLIST[I]);
: 2349      2471 2
: 2350      2472 2
: 2351      2473 2      ! Mark this symbol's RST entry as being referenced by adding its address
: 2352      2474 2      ! to the RST Reference List (RST$REF_LIST). This only says that the RST
: 2353      2475 2      ! entry is referenced by the current Debug command. Note that we expand
: 2354      2476 2      ! the list memory block if it is about to overflow.
: 2355      2477 2
: 2356      2478 2      IF .RST$REF_LIST[1] EQL .RST$REF_LIST[0]
: 2357      2479 2      THEN
: 2358      2480 2          BEGIN
: 2359      2481 2              RST$REF_LIST[0] = .RST$REF_LIST[0] + 20;
: 2360      2482 2              NEWREFLIST = DBG$GET_MEMORY(.RST$REF_LIST[0] + 2);
: 2361      2483 2              CH$MOVE(4*(.RST$REF_LIST[1] + 2), .RST$REF_LIST, .NEWREFLIST);
: 2362      2484 2              DBG$REL_MEMORY(.RST$REF_LIST);
: 2363      2485 2              RST$REF_LIST = .NEWREFLIST;
: 2364      2486 2          END;
: 2365      2487 2
: 2366      2488 2      RST$REF_LIST[1] = .RST$REF_LIST[1] + 1;
: 2367      2489 2      RST$REF_LIST[.RST$REF_LIST[1] + 1] = .RSTPTR;
: 2368      2490 2
: 2369      2491 2
: 2370      2492 2      ! Mark the symbol's module as being the Most Recently Referenced module.
: 2371      2493 2      ! Then return.
: 2372      2494 2
: 2373      2495 2      IF .MODRSTPTR NEQ .LRUM$MOST_RECENT THEN DBG$RST_MOST_RECENT(.MODRSTPTR);
: 2374      2496 2      RETURN;
: 2375      2497 2
: 2376      2498 1      END;
```

```
53 54 45 47 5C 53 53 45 43 43 20 45 4E 49 4C 25 00082 P.AAN: .ASCII \XLINE \
00000000 00000000 00000003 00000000 00088 P.AAO: .LONG 0, 3, 0, 0
41 54 53 52 13 00098 P.AAP: .ASCII <19>\RSTACCESS\<92>\GETSYMBOL\
4C 4F 42 4D 59 000A7
```

.PSECT DBG\$OWN,NOEXE, PIC,2

```
00000000 00000000 00000001 00000000 0001A .BLKB 2
00000000 00000000 00000000 0001C CANDLIST: .LONG 0
00000000 00000000 00000001 00000000 00020 MODU_SCOPE:
.LONG 0, 1, 0, 0
00000000 00000000 00000001 00000000 00030 NORM_SCOPE:
.LONG 0, 1, 0, 0
00000000 00000000 00000002 00000000 00040 NUMB_SCOPE:
.LONG 0, 2, 0, 0
```

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

```
OFFC 00000 .ENTRY DBG$STA GETSYMBOL, Save R2,R3,R4,R5,R6,R7,- ; 1159
SE FC40 CE 9E 00002 MOVAB R8,R9,R10,R11
-960(SP), SP
```

	1C	AE	00000000G	00	D0	00007	MOVL	RST\$SET_SCOPE, SET_SCOPE	1344	
			00000000G	00	D4	0000F	CLRL	RST\$SET_SCOPE	1345	
57	04	AC		08	C1	00015	ADDL3	#8, PATHNAME, PATHVEC	1351	
	56		04	BC	9A	0001A	MOVZBL	@PATHNAME, R6	1352	
	18	AE	FC	A746	D0	0001E	MOVL	-4(PATHVEC)[R6], NAMEPTR		
			10	AE	7C	00024	CLRL	LINE_NUM_IS_LAST	1359	
			0C	AE	D4	00027	CLRL	LINE_NUM_LOC	1360	
	58			01	D0	0002A	MOVL	#1, VALID_LINE_FLAG	1361	
				55	D4	0002D	CLRL	I	1362	
			00B8	31	0002F	1\$:	BRW	12\$		
	08	AE	FC	A745	D0	00032	2\$:	MOVL	-4(PATHVEC)[I], PNAME	1364
	59		08	BE	9A	00038	MOVZBL	@PNAME, R9	1365	
	06			59	91	0003C	CMPB	R9, #6		
				EE	1B	0003F	BLEQU	1\$		
00000000*	5A	08	AE	01	C1	00041	ADDL3	#1, PNAME, R10	1366	
	EF	6A		06	29	00046	CMPC3	#6, (R10), P.AAN		
				DF	12	0004E	BNEQ	1\$		
50	08	AE		07	C1	00050	ADDL3	#7, PNAME, R0	1369	
	30			60	91	00055	CMPB	(R0), #48		
				0A	1F	00058	BLSSU	3\$		
50	08	AE		07	C1	0005A	ADDL3	#7, PNAME, R0		
	39			60	91	0005F	CMPB	(R0), #57		
				02	1B	00062	BLEQU	4\$		
				58	D4	00064	3\$:	CLRL	VALID_LINE_FLAG	
	02	14	AE	E9	00066	4\$:	BLBC	HAVE [LINE_NUM, 5\$	1370	
				58	D4	0006A	CLRL	VALID_LINE_FLAG		
	14	AE		01	D0	0006C	5\$:	MOVL	#1, HAVE_LINE_NUM	1371
	0C	AE		55	D0	00070	MOVL	I, LINE_NUM_LOC	1372	
	04	AE		01	CE	00074	MNEGL	#1, LINE_NUM	1378	
				54	D4	00078	CLRL	NUMBER	1379	
	50			06	D0	0007A	MOVL	#6, I	1380	
				52	11	0007D	BRB	9\$		
	51	08	AE	D0	0007F	6\$:	MOVL	PNAME, R1	1382	
	2E		6041	91	00083		CMPB	(I)[R1], #46		
				12	12	00087	BNEQ	7\$		
FFFFFFFF	8F	04	AE	D1	00089		CMPL	LINE_NUM, #-1		
				08	12	00091	BNEQ	7\$		
	04	AE		54	D0	00093	MOVL	NUMBER, LINE_NUM	1385	
				54	D4	00097	CLRL	NUMBER	1386	
				36	11	00099	BRB	9\$	1382	
	51	08	AE	D0	0009B	7\$:	MOVL	PNAME, R1	1389	
	30		6041	91	0009F		CMPB	(I)[R1], #48		
				28	1F	000A3	BLSSU	8\$		
	51	08	AE	D0	000A5		MOVL	PNAME, R1		
	39		6041	91	000A9		CMPB	(I)[R1], #57		
				1E	1A	000AD	BGTRU	8\$		
000F4240	8F			54	D1	000AF	CMPL	NUMBER, #1000000	1390	
				15	14	000B6	BGTR	8\$		
51	54			0A	C5	000B8	MULL3	#10, NUMBER, R1	1392	
	53	08	AE	D0	000BC		MOVL	PNAME, R3		
	52		6043	9A	000C0		MOVZBL	(I)[R3], R2		
	51			52	C0	000C4	ADDL2	R1		
	54	D0	A1	9E	000C7		MOVAB	-48(R1), NUMBER		
				04	11	000CB	BRB	9\$		
				58	D4	000CD	8\$:	CLRL	VALID_LINE_FLAG	1396
				04	11	000CF	BRB	10\$	1395	
AA	50			59	F3	000D1	9\$:	AOBLEQ	R9, I, 6\$	1380

			FFFFFFF	8F	04	AE	D1	000D5	10\$:	CMPL	LINE_NUM, #-1	1405	
						08	12	000DD		BNEQ	11\$		
			04	AE		54	D0	000DF		MOVL	NUMBER, LINE_NUM	1408	
						6E	D4	000E3		CLRL	STMT_NUM	1409	
						03	11	000E5		BRB	12\$	1405	
						54	D0	000E7	11\$:	MOVL	NUMBER, STMT_NUM	1413	
FF42		55		6E		01	F1	000EA	12\$:	ACBL	R6, #1, 1, 2\$	1362	
				3A	14	AE	E9	000F0		BLBC	HAVE_LINE_NUM, 16\$	1424	
						50	D4	000F4		CLRL	R0	1427	
				56	0C	AE	D1	000F6		CMPL	LINE_NUM_LOC, R6		
						02	12	000FA		BNEQ	13\$		
						50	D6	000FC		INCL	R0		
			10	AE		50	D0	000FE	13\$:	MOVL	R0, LINE_NUM_IS_LAST		
OC	AE	04	BC	08		08	ED	00102		CMPZV	#8, #8, @PATHNAME, LINE_NUM_LOC	1428	
						1B	19	00109		BLSS	14\$		
				08		08	EF	0010B		EXTZV	#8, #8, @PATHNAME, R0	1429	
						50	D7	00111		DECL	R0		
				50	0C	AE	D1	00113		CMPL	LINE_NUM_LOC, R0		
						0D	19	00117		BLSS	14\$		
OC	AE	04	BC	08		08	ED	00119		CMPZV	#8, #8, @PATHNAME, LINE_NUM_LOC	1430	
						06	12	00120		BNEQ	15\$		
				02	10	AE	E8	00122		BLBS	LINE_NUM_IS_LAST, 15\$		
						58	D4	00126	14\$:	CLRL	VALID_LINE_FLAG	1432	
				03		58	E8	00128	15\$:	BLBS	VALID_LINE_FLAG, 16\$	1434	
						0234	31	0012B		BRW	40\$		
						58	D4	0012E	16\$:	CLRL	NCANDS	1448	
				00000000'		EF	D5	00130		TSTL	CANDLST	1449	
						17	12	00136		BNEQ	17\$		
						08	DD	00138		PUSHL	#11	1452	
				00000000G	00	01	FB	0013A		CALLS	#1, DBG\$GET_MEMORY		
				00000000'	EF	50	D0	00141		MOVL	R0, CANDLST		
				00000000'	FF	0A	D0	00148		MOVL	#10, @CANDLST	1453	
					5B	00000000G	00	D0	0014F	17\$:	MOVL	SCOPE\$LIST, SCOPEPTR	1462
					60	AE	D4	00156		CLRL	HAVE_NUM_SCOPE	1463	
				08	AE	67	50	00159		MOVL	(PATRVECT, PNAME	1464	
					08	BE	95	0015D		TSTB	@PNAME	1465	
						51	12	00160		BNEQ	21\$		
00		04	BC	08		10	ED	00162		CMPZV	#16, #8, @PATHNAME, #0	1468	
						09	12	00168		BNEQ	18\$		
				5B	00000000'	EF	9E	0016A		MOVAB	P.AAO, SCOPEPTR	1470	
						3D	11	00171		BRB	20\$		
01		04	BC	08		10	ED	00173	18\$:	CMPZV	#16, #8, @PATHNAME, #1	1472	
						20	12	00179		BNEQ	19\$		
				60	AE	01	D0	0017B		MOVL	#1, HAVE_NUM_SCOPE	1475	
				5B	00000000'	EF	9E	0017F		MOVAB	NUMB_SCOPE, SCOPEPTR	1476	
						04	C1	00186		ADDL3	#4, PATHNAME, R0	1477	
			50	04	AC	60	D0	0018B		MOVL	(R0), 12(SCOPEPTR)		
02		04	BC	08	0C	08	ED	0018F		CMPZV	#8, #8, @PATHNAME, #2	1478	
						19	18	00195		BGEQ	20\$		
						58	D4	00197		CLRL	SCOPEPTR		
						15	11	00199		BRB	20\$	1472	
				00000000'		EF	9F	0019B	19\$:	PUSHAB	P.AAP	1482	
						01	DD	001A1		PUSHL	#1		
				00028362		8F	DD	001A3		PUSHL	#164706		
				00000000G	00	03	FB	001A9		CALLS	#3, LIB\$SIGNAL		
						0140	31	001B0	20\$:	BRW	36\$	1465	
01		04	BC	08		08	ED	001B3	21\$:	CMPZV	#8, #8, @PATHNAME, #1	1491	

54	04	BC	01	OC	F5	15	001B9	BLEQ	20\$		
					AE	D1	001BB	CMPL	LINE_NUM_LOC, #1		
					EF	13	001BF	BEQL	20\$		
			08		08	EF	001C1	EXTZV	#8, #8, @PATHNAME, PATH_START_LOC		1494
					54	D7	001C7	DECL	PATH_START_LOC		
			54	OC	AE	D1	001C9	CMPL	LINE_NUM_LOC, PATH_START_LOC		1495
					02	12	001CD	BNEQ	22\$		
					54	D7	001CF	DECL	PATH_START_LOC		1497
				00000000'	EF	D4	001D1	CLRL	MODU_SCOPE+8		1504
				00000000'	EF	D4	001D7	CLRL	NORM_SCOPE+8		1505
			55	FC	A744	D0	001DD	MOVL	-4(PATHVEC)[PATH_START_LOC], PATH_NAME_PTR		1506
					55	DD	001E2	PUSHL	PATH_NAME_PTR		1507
		00000000G	00		01	FB	001E4	CALLS	#1, DBG\$HASH_FIND_SETUP		
					55	DD	001EB	PUSHL	PATH_NAME_PTR		1514
		00000000G	00		01	FB	001ED	CALLS	#1, DBG\$HASH_FIND		
		20	AE		50	D0	001F4	MOVL	R0, RSTPTR		
					03	12	001F8	BNEQ	24\$		1515
					00C5	31	001FA	BRW	34\$		
			59	20	AE	D0	001FD	MOVL	RSTPTR, RPTR		1521
		24	AE		54	D0	00201	MOVL	PATH_START_LOC, PINDEX		1522
	50	20	AE		15	C1	00205	ADDL3	#21, RSTPTR, R0		1525
			DE		60	E8	0020A	BLBS	(R0), 23\$		
	50	20	AE		14	C1	0020D	ADDL3	#20, RSTPTR, R0		1526
			01		60	91	00212	CMPB	(R0), #1		
					1E	13	00215	BEQL	26\$		
	50	20	AE		14	C1	00217	ADDL3	#20, RSTPTR, R0		1527
			02		60	91	0021C	CMPB	(R0), #2		
					14	13	0021F	BEQL	26\$		
	50	20	AE		14	C1	00221	ADDL3	#20, RSTPTR, R0		1528
			03		60	91	00226	CMPB	(R0), #3		
					0A	13	00229	BEQL	26\$		
	50	20	AE		14	C1	0022B	ADDL3	#20, RSTPTR, R0		1529
			07		60	91	00230	CMPB	(R0), #7		
					B6	12	00233	BNEQ	23\$		
			50	24	AE	D0	00235	MOVL	PINDEX, R0		1533
		08	AE	FC	A740	D0	00239	MOVL	-4(PATHVEC)[R0], PNAME		1534
				OC	A9	DD	0023F	PUSHL	12(RPTR)		
		00000000G	00		01	FB	00242	CALLS	#1, DBG\$GET_DST_NAME		
		3C	AE		50	D0	00249	MOVL	R0, RNAME		
			51	08	BE	9A	0024D	MOVZBL	@PNAME, R1		1535
			50	3C	BE	9A	00251	MOVZBL	@RNAME, R0		
	56	3C	AE		01	C1	00255	ADDL3	#1, RNAME, R6		
	5A	08	AE		01	C1	0025A	ADDL3	#1, PNAME, R10		
	50	00	6A		51	2D	0025F	CMPC5	R1, (R10), #0, R0, (R6)		
					66		00264				
			01	24	AE	D1	00267	BNEQ	33\$		1544
					45	12	0026B	CMPL	PINDEX, #1		
		0090	CE	20	AE	D0	0026D	BNEQ	32\$		
			50	0090	CE	D0	00273	MOVL	RSTPTR, MODRSTPTR		1547
			01	14	A0	91	00278	MOVL	MODRSTPTR, R0		1548
					08	13	0027C	CMPB	20(R0), #1		
		0090	CE	10	A0	D0	0027E	BEQL	28\$		
					ED	11	00284	MOVL	16(R0), MODRSTPTR		1549
			5B	00000000'	EF	9E	00286	BRB	27\$		
		0090	CE	20	AE	D1	0028D	MOVAB	NORM_SCOPE, SCOPEPTR		1551
					07	12	00293	CMPL	RSTPTR, MODRSTPTR		1552
								BNEQ	29\$		

	5B	00000000'	EF	9E	00295	MOVAB	MODU_SCOPE, SCOPEPTR		
		08	AB	D5	0029C	29\$: TSTL	8(SCOPEPTR)		1553
			03	13	0029F	BEQL	30\$		
			06BC	31	002A1	BRW	130\$		
08	AB	20	AE	D0	002A4	30\$: MOVL	RSTPTR, 8(SCOPEPTR)		1561
0C	AB	0090	CE	D0	002A9	MOVL	MODRSTPTR, 12(SCOPEPTR)		1562
			FF39	31	002AF	31\$: BRW	23\$		1546
		24	AE	D7	002B2	32\$: DECL	PINDEX		1569
01		14	A9	91	002B5	33\$: CMPB	20(RPTR), #1		1575
			F4	13	002B9	BEQL	31\$		
59		10	A9	D0	002BB	MOVL	16(RPTR), RPTR		1576
			FF43	31	002BF	BRW	25\$		1523
			5B	D4	002C2	34\$: CLRL	SCOPEPTR		1586
		00000000'	EF	D4	002C4	CLRL	MODU_SCOPE		1587
		00000000'	EF	D5	002CA	TSTL	NORM_SCOPE+8		1588
			12	13	002D0	BEQL	35\$		
00000000'	EF	00000000'	EF	9E	002D2	MOVAB	NORM_SCOPE, MODU_SCOPE		1591
	5B	00000000'	EF	9E	002DD	MOVAB	NORM_SCOPE, SCOPEPTR		1592
		00000000'	EF	D5	002E4	35\$: TSTL	MODU_SCOPE+8		1595
			07	13	002EA	BEQL	36\$		
	5B	00000000'	EF	9E	002EC	MOVAB	MODU_SCOPE, SCOPEPTR		
28	AE	00000000G	00	D0	002F3	36\$: MOVL	RST\$START_ADDR, NEXTSETMOD		1603
		74	AE	7C	002FB	CLRQ	REG_SCOPE		1604
		64	AE	D4	002FE	CLRL	FIRST_MODPTR		1606
6C	AE		5B	D0	00301	MOVL	SCOPEPTR, SCOPE_START_PTR		1607
	50	00000000'	EF	9E	00305	MOVAB	MODU_SCOPE, R0		1608
	50		5B	D1	0030C	CMPL	SCOPEPTR, R0		
		00000000'	10	12	0030F	BNEQ	37\$		
			EF	D5	00311	TSTL	MODU_SCOPE		
			08	13	00317	BEQL	37\$		
6C	AE	00000000'	EF	D0	00319	MOVL	MODU_SCOPE, SCOPE_START_PTR		1610
		5C	AE	D4	00321	37\$: CLRL	HAVE_SCOPE		1623
			5B	D5	00324	38\$: TSTL	SCOPEPTR		1635
			03	13	00326	BEQL	39\$		
			00BE	31	00328	BRW	45\$		
		78	AE	DD	0032B	39\$: PUSHL	REG_LINE_LEX_PTR		1643
		70	AE	DD	0032E	PUSHL	SCOPE_START_PTR		
		04	AC	DD	00331	PUSHL	PATHNAME		1642
0000V	CF		03	FB	00334	CALLS	#3, GET_REGISTER_SYMID		
54	AE		50	D0	00339	MOVL	R0, REGISTER_SYMID		
			2A	12	0033D	BNEQ	41\$		1651
	1F	14	AE	E9	0033F	BLBC	HAVE_LINE_NUM, 40\$		1654
		64	AE	D5	00343	TSTL	FIRST_MODPTR		
			1A	13	00346	BEQL	40\$		
			01	DD	00348	PUSHL	#1		1656
		0098	CE	9F	0034A	PUSHAB	LINEEND		
		00A0	CE	9F	0034E	PUSHAB	LINESTART		
		70	AE	DD	00352	PUSHL	FIRST_MODPTR		1657
		10	AE	DD	00355	PUSHL	STMT_NUM		1656
		18	AE	DD	00358	PUSHL	LINE_NUM		
00000000G	00		06	FB	0035B	CALLS	#6, DBG\$LINE_TO_PC_LOOKUP		
		08	BC	D4	00362	40\$: CLRL	@SYMID		1659
		0C	BC	D4	00365	CLRL	@KIND		1660
			04	00368	RET				1653
	08	BC	54	AE	D0	41\$: MOVL	REGISTER_SYMID, @SYMID		1668
50	54	AE	14	C1	0036E	ADDL3	#20, REGISTER_SYMID, R0		1669
	0C	BC	60	9A	00373	MOVZBL	(R0), @KIND		

			10	AC	D5	00377	TSTL	OUT_SCOPE_STATE	1675	
			03	13	0037A	BEQL	42\$			
			10	BC	D4	0037C	CLRL	@OUT_SCOPE_STATE	1677	
			14	AC	D5	0037F	TSTL	OUT_SCOPE	1678	
			03	13	00382	BEQL	43\$			
			14	BC	D4	00384	CLRL	@OUT_SCOPE	1680	
		50	00000000G	00	D0	00387	MOVL	RST\$REF_LIST, R0	1688	
		60	04	A0	D1	0038E	CMPL	4(R0), 7(R0)		
				40	12	00392	BNEQ	44\$		
		60		14	C0	00394	ADDL2	#20, (R0)	1691	
7E		60		02	C1	00397	ADDL3	#2, (R0), -(SP)	1692	
	00000000G	00		01	FB	00398	CALLS	#1, DBG\$GET_MEMORY		
	7C	AE		50	D0	003A2	MOVL	R0, NEWREFLIST		
	40	AE	00000000G	00	D0	003A6	MOVL	RST\$REF_LIST, 64(SP)	1693	
51	40	AE		04	C1	003AE	ADDL3	#4, 64(SP), R1		
		50		61	D0	003B3	MOVL	(R1), R0		
		50		04	C4	003B6	MULL2	#4, R0		
		50		08	C0	003B9	ADDL2	#8, R0		
70	BE	40	BE	50	28	003BC	MOVC3	R0, @64(SP), @NEWREFLIST		
			40	AE	DD	003C2	PUSHL	64(SP)	1694	
	00000000G	00		01	FB	003C5	CALLS	#1, DBG\$REL_MEMORY		
	00000000G	00		70	AE	003CC	MOVL	NEWREFLIST, RST\$REF_LIST	1695	
		50	00000000G	00	D0	003D4	MOVL	RST\$REF_LIST, R0	1698	
			04	A0	D6	003DB	INCL	4(R0)		
	51		04	A0	D0	003DE	MOVL	4(R0), R1	1699	
	04	A041	20	AE	D0	003E2	MOVL	RSTPTR, 4(R0)[R1]		
					04	003E8	RET		1637	
			50	D4	003E9	45\$:	CLRL	R0	1707	
	6C	AE		5B	D1	003EB	CMPL	SCOPEPTR, SCOPE_START_PTR		
				02	12	003EF	BNEQ	46\$		
			50	D6	003F1		INCL	R0		
	74	AE		50	D0	003F3	46\$:	MOVL	R0, REG_SCOPE	
	50	AE	04	AB	D0	003F7	MOVL	4(SCOPEPTR), SCOPE_STATE	1712	
008F	03	01	50	AE	CF	003FC	CASEL	SCOPE_STATE, #1, #3	1713	
	003A	0019	0008		00401	47\$:	.WORD	48\$-47\$,-		
								49\$-47\$,-		
								52\$-47\$,-		
								57\$-47\$		
	008C	CE	08	AB	7D	00409	48\$:	MOVQ	8(SCOPEPTR), SCOPE	1724
		50	0090	CE	D0	0040F	MOVL	MODRSTPTR, R0	1726	
		20	28	A0	E9	00414	BLBC	40(R0), 51\$		
				1A	11	00418	BRB	50\$		
			0088	CE	9F	0041A	49\$:	PUSHAB	NUMSCP_INVOC_NUM	1739
			0090	CE	9F	0041E	PUSHAB	SCOPE		
			0098	CE	9F	00422	PUSHAB	MODRSTPTR		
			0C	AB	DD	00426	PUSHL	12(SCOPEPTR)		
	0000V	CF	008C	CE	D5	0042E	CALLS	#4, DBG\$STA_NUMBERED_SCOPE		
				04	FB	00429	TSTL	SCOPE	1741	
				04	13	00432	BEQL	51\$		
	5C	AE		01	D0	00434	50\$:	MOVL	#1, HAVE_SCOPE	
			00AB	31	00438	51\$:	BRW	63\$	1742	
	08	AE		67	D0	0043B	52\$:	MOVL	(PATHVEC), PNAME	1753
		50	04	BC	9A	0043F	MOVZBL	@PATHNAME, R0	1754	
50	04	BC		08	ED	00443	CMPZV	#8, #8, @PATHNAME, R0		
				ED	12	00449	BNEQ	51\$		
		02	04	BC	91	0044B	CMPB	@PATHNAME, #2	1755	
				05	12	0044F	BNEQ	53\$		

			08	BE	95	00451	TSTB	@PNAME			
			06	13	00454		BEQL	54\$			
		01	04	BC	91	00456	53\$:	CMPB	@PATHNAME, #1	1756	
				DC	12	0045A		BNEQ	51\$		
		50	04	BC	9A	0045C	54\$:	MOVZBL	@PATHNAME, R0	1760	
			FC	A740	DD	00460		PUSHL	-4(PATHVEC)[R0]		
	0000V	CF		01	FB	00464		CALLS	#1, DBG\$STA_LOOKUP_GBL		
	20	AE		50	00	00469		MOVL	R0, RSTPTR		
				77	13	0046D		BEQL	63\$	1761	
50	08	BC	20	AE	D0	0046F		MOVL	RSTPTR, @SYMID	1764	
	20	AE		14	C1	00474		ADDL3	#20, RSTPTR, R0	1765	
	0C	BC		60	9A	00479		MOVZBL	(R0), @KIND		
			10	AC	D5	0047D		TSTL	OUT_SCOPE_STATE	1770	
				04	13	00480		BEQL	55\$		
	10	BC		03	D0	00482		MOVL	#3, @OUT_SCOPE_STATE	1772	
			14	AC	D5	00486	55\$:	TSTL	OUT_SCOPE	1773	
				01	12	00489		BNEQ	56\$		
					04	0048B		RET			
			14	BC	D4	0048C	56\$:	CLRL	@OUT_SCOPE	1775	
					04	0048F		RET		1763	
	00000000G	00	28	AE	D1	00490	57\$:	CMPB	NEXTSETMOD, RST\$START_ADDR	1798	
				18	12	00498		BNEQ	59\$		
			28	AE	D5	0049A	58\$:	TSTL	NEXTSETMOD	1801	
				13	13	0049D		BEQL	59\$		
50	28	AE		28	C1	0049F		ADDL3	#40, NEXTSETMOD, R0	1803	
		0B		60	E8	004A4		BLBS	(R0), 59\$		
50	28	AE		10	C1	004A7		ADDL3	#16, NEXTSETMOD, R0	1804	
	28	AE		60	D0	004AC		MOVL	(R0), NEXTSETMOD		
				E8	11	004B0		BRB	58\$	1801	
	0090	CE	28	AE	D0	004B2	59\$:	MOVL	NEXTSETMOD, MODRSTPTR	1815	
	008C	CE	0090	CE	D0	004B8		MOVL	MODRSTPTR, SCOPE	1816	
			28	AE	D5	004BF	60\$:	TSTL	NEXTSETMOD	1817	
				13	13	004C2		BEQL	61\$		
50	28	AE		10	C1	004C4		ADDL3	#16, NEXTSETMOD, R0	1819	
	28	AE		60	D0	004C9		MOVL	(R0), NEXTSETMOD		
				08	13	004CD		BEQL	61\$	1820	
50	28	AE		28	C1	004CF		ADDL3	#40, NEXTSETMOD, R0	1821	
		E8		60	E9	004D4		BLBC	(R0), 60\$		
			0090	CE	D5	004D7	61\$:	TSTL	MODRSTPTR	1824	
				04	13	004DB		BEQL	62\$		
	5C	AE		01	D0	004DD		MOVL	#1, HAVE_SCOPE		
			28	AE	D5	004E1	62\$:	TSTL	NEXTSETMOD	1825	
				03	12	004E4		BNEQ	64\$		
		5B		6B	D0	004E6	63\$:	MOVL	(SCOPEPTR), SCOPEPTR		
		03	5C	AE	E8	004E9	64\$:	BLBS	HAVE_SCOPE, 65\$	1834	
				FE34	31	004ED		BRW	38\$		
		52	0090	CE	D0	004F0	65\$:	MOVL	MODRSTPTR, R2	1842	
0B	28	A2		01	E0	004F5		BBS	#1, 40(R2), 66\$		
				7E	D4	004FA		CLRL	-(SP)	1844	
				52	DD	004FC		PIJSHL	R2		
	00000000G	00		02	FB	004FE		CALLS	#2, DBG\$RST_BUILD		
			10	AC	D5	00505	66\$:	TSTL	OUT_SCOPE_STATE	1850	
				05	13	00508		BEQL	67\$		
	10	BC	50	AE	D0	0050A		MOVL	SCOPE_STATE, @OUT_SCOPE_STATE	1852	
			14	AC	D5	0050F	67\$:	TSTL	OUT_SCOPE	1853	
				06	13	00512		BEQL	68\$		
	14	BC	008C	CE	D0	00514		MOVL	SCOPE, @OUT_SCOPE	1855	

	03	14	AE	E8	0051A	68\$:	BLBS	HAVE_LINE_NUM, 69\$	1862
		00A3	31	0051E			BRW	79\$	
		64	AE	D5	00521	69\$:	TSTL	FIRST_MODPTR	1871
			04	12	00524		BNEQ	70\$	
	64	AE	52	D0	00526		MOVL	R2, FIRST_MODPTR	
			7E	D4	0052A	70\$:	CLRL	-(SP)	1876
		0098	CE	9F	0052C		PUSHAB	LINEEND	
		00A0	CE	9F	00530		PUSHAB	LINESTART	
			52	DD	00534		PUSHL	R2	1877
		10	AE	DD	00536		PUSHL	STMT_NUM	1876
		18	AE	DD	00539		PUSHL	LINE_NUM	
00000000G	00		06	FB	0053C		CALLS	#6, DBG\$LINE_TO_PC_LOOKUP	
0084	CE		50	D0	00543		MOVL	R0, STATUS	
38	AE	18	A2	D0	00548		MOVL	24(R2), SATPTR	1884
	03	0084	CE	E8	0054D		BLBS	STATUS, 71\$	1885
		38	AE	D4	00552		CLRL	SATPTR	
		48	AE	D4	00555	71\$:	CLRL	LINE_LEX_PTR	1886
		38	AE	D5	00558	72\$:	TSTL	SATPTR	1887
			5E	13	0055B		BEQL	78\$	
50	38	AE	04	C1	0055D		ADDL3	#4, SATPTR, R0	1889
	0098	CE	60	D1	00562		CMPL	(R0), LINESTART	
			52	14	00567		BGTR	78\$	
50	38	AE	0C	C1	00569		ADDL3	#12, SATPTR, R0	1890
	20	AE	60	D0	0056E		MOVL	(R0), RSTPTR	
50	38	AE	08	C1	00572		ADDL3	#8, SATPTR, R0	1891
	0098	CE	60	D1	00577		CMPL	(R0), LINESTART	
			36	19	0057C		BLSS	77\$	
50	20	AE	14	C1	0057E		ADDL3	#20, RSTPTR, R0	1892
	02		60	91	00583		CPMB	(R0), #2	
			0A	13	00586		BEQL	73\$	
50	20	AE	14	C1	00588		ADDL3	#20, RSTPTR, R0	1893
	03		60	91	0058D		CPMB	(R0), #3	
			22	12	00590		BNEQ	77\$	
		48	AE	D5	00592	73\$:	TSTL	LINE_LEX_PTR	1896
			10	13	00595		BEQL	75\$	
	59	20	AE	D0	00597		MOVL	RSTPTR, RPTR	1902
	01	14	A9	91	0059B	74\$:	CPMB	20(RPTR), #1	1903
			13	13	0059F		BEQL	77\$	
48	AE		59	D1	005A1		CMPL	RPTR, LINE_LEX_PTR	1905
			07	12	005A5		BNEQ	76\$	
48	AE	20	AE	D0	005A7	75\$:	MOVL	RSTPTR, LINE_LEX_PTR	1908
			06	11	005AC		BRB	77\$	1907
	59	10	A9	D0	005AE	76\$:	MOVL	16(RPTR), RPTR	1912
			E7	11	005B2		BRB	74\$	1903
38	AE	38	BE	D0	005B4	77\$:	MOVL	@SATPTR, SATPTR	1919
			9D	11	005B9		BRB	72\$	1887
	05	74	AE	E9	005BB	78\$:	BLBC	REG SCOPE, 79\$	1927
78	AE	48	AE	D0	005BF		MOVL	LINE_LEX_PTR, REG_LINE_LEX_PTR	
		18	AE	DD	005C4	79\$:	PUSHL	NAMEPTR	1937
00000000G	00		01	FB	005C7		CALLS	#1, DBG\$HASH_FIND_SETUP	
	0A	10	AE	E9	005CE	80\$:	BLBC	LINE_NUM_IS_CAST, 81\$	1946
20	AE	48	AE	D0	005D2		MOVL	LINE_LEX_PTR, RSTPTR	1949
		48	AE	D4	005D7		CLRL	LINE_LEX_PTR	1950
			0E	11	005DA		BRB	82\$	1946
		18	AE	DD	005DC	81\$:	PUSHL	NAMEPTR	1958
00000000G	00		01	FB	005DF		CALLS	#1, DBG\$HASH_FIND	
20	AE	50	D0	005E6			MOVL	R0, RSTPTR	

			20	AE	D5	005EA	82\$:	TSTL	RSTPTR	1964
				03	12	005ED		BNEQ	83\$	
				02BA	31	005EF		BRW	116\$	
				56	D4	005F2	83\$:	CLRL	STKPTR	1971
		59	20	AE	D0	005F4		MOVL	RSTPTR, RPTR	1972
	24	AE	04	BC	9A	005F8		MOVZBL	@PATHNAME, PINDEX	1973
50	20	AE		14	C1	005FD		ADDL3	#20, RSTPTR, R0	1981
	40	AE		60	9E	00602		MOVAB	(R0), 64(SP)	
C3	40	BE		08	E0	00606	84\$:	BBS	#8, @64(SP), 80\$	
		01	40	BE	91	0060B		CMPB	@64(SP), #1	1982
				BD	13	0060F		BEQL	80\$	
		06	1C	AC	E8	00611		BLBS	TYPE FLAG, 85\$	1983
		07	40	BE	91	00615		CMPB	@64(SP), #7	
				B3	13	00619		BEQL	80\$	
		0C	1C	AE	E9	0061B	85\$:	BLBC	SET SCOPE, 86\$	1991
		02	40	BE	91	0061F		CMPB	@64(SP), #2	1994
				06	13	00623		BEQL	86\$	
		03	40	BE	91	00625		CMPB	@64(SP), #3	1995
				A3	12	00629		BNEQ	80\$	
	00000064	8F		56	D6	0062B	86\$:	INCL	STKPTR	2005
				56	D1	0062D		CMPL	STKPTR, #100	2006
				98	18	00634		BGEQ	80\$	
			00A0	CE46	7F	00636		PUSHAQ	SYMSTACK[STKPTR]	2007
		9E		59	D0	0063B		MOVL	RPTR, @ (SP)+	
		54	00A4	CE46	7E	0063E		MOVAQ	SYMSTACK+4[STKPTR], R4	2008
				64	D4	00644		CLRL	(R4)	
		07	14	AE	E9	00646		BLBC	HAVE LINE NUM, 87\$	2015
	0C	AE	24	AE	D1	0064A		CMPL	PINDEX, LINE_NUM_LOC	
				0A	13	0064F		BEQL	88\$	
		09	60	AE	E9	00651	87\$:	BLBC	HAVE NUM SCOPE, 89\$	2016
		01	24	AE	D1	00655		CMPL	PINDEX, #1	
				03	12	00659		BNEQ	89\$	
			24	AE	D7	0065B	88\$:	DECL	PINDEX	2018
		50	24	AE	D0	0065E	89\$:	MOVL	PINDEX, R0	2026
	08	AE	FC	A740	D0	00662		MOVL	-4(PATHVEC)[R0], PNAME	
				55	D4	00668		CLRL	R5	2027
			24	AE	D5	0066A		TSTL	PINDEX	
				06	12	0066D		BNEQ	90\$	
				55	D6	0066F		INCL	R5	
	08	AE		67	D0	00671		MOVL	(PATHVEC), PNAME	
			0C	A9	DD	00675	90\$:	PUSHL	12(RPTR)	2028
	00000000G	00		01	FB	00678		CALLS	#1, DBG\$GET_DST_NAME	
	3C	AE		50	D0	0067F		MOVL	R0, RNAME	
		51	08	BE	9A	00683		MOVZBL	@PNAME, R1	2029
		50	3C	BE	9A	00687		MOVZBL	@RNAME, R0	
				01	C1	0068B		ADDL3	#1, RNAME, -(SP)	
	7E	3C		01	C1	00690		ADDL3	#1, PNAME, -(SP)	
	7E	0C								
50	00	9E		51	2D	00695		CMPC5	R1, @ (SP)+, #0, R0, @ (SP)+	
				9E		0069A				
				06	13	0069B		BEQL	91\$	
		03		55	E8	0069D		BLBS	R5, 91\$	2030
				01C1	31	006A0		BRW	112\$	
		64	24	AE	B0	006A3	91\$:	MOVW	PINDEX, (R4)	2037
24	AE	04	BC	08	ED	006A7		CMPZV	#8, #8, @PATHNAME, PINDEX	2044
				03	18	006AE		BGEQ	93\$	
				01AA	31	006B0	92\$:	BRW	111\$	
		0A	14	A9	91	006B3	93\$:	CMPB	20(RPTR), #10	2045

	34	AE	20	F7	13	006B7	BEQL	92\$		
		01	40	AE	D0	006B9	MOVL	RSTPTR, SYSCOPE	2052	
				BE	91	006BE	CMPB	@64(SP), #1	2053	
				09	13	006C2	BEQL	94\$		
50	20	AE		10	C1	006C4	ADDL3	#16, RSTPTR, R0	2055	
	34	AE		60	D0	006C9	MOVL	(R0), SYMSCOPE		
50	34	AE		14	C1	006CD	ADDL3	#20, SYMSCOPE, R0	2057	
		07		60	91	006D2	CMPB	(R0), #7		
				09	12	006D5	BNEQ	95\$		
50	34	AE		10	C1	006D7	ADDL3	#16, SYMSCOPE, R0	2059	
	34	AE		60	D0	006DC	MOVL	(R0), SYMSCOPE		
		04	50	AE	D1	006E0	CMPL	SCOPE_STATE, #4	2068	
				04	13	006E4	BEQL	96\$		
		15	10	AE	E9	006E6	BLBC	LINE_NUM_IS_LAST, 97\$		
50	34	AE		14	C1	006EA	ADDL3	#20, SYMSCOPE, R0	2071	
		01		60	91	006EF	CMPB	(R0), #1		
				0B	13	006F2	BEQL	97\$		
50	34	AE		10	C1	006F4	ADDL3	#16, SYMSCOPE, R0	2072	
	34	AE		60	D0	006F9	MOVL	(R0), SYMSCOPE		
				EB	11	006FD	BRB	96\$		
	2C	AE	008C	CE	D0	006FF	MOVL	SCOPE, SCPTR	2079	
			0080	CE	D4	00705	CLRL	DEFDEPTH	2080	
	58	AE		01	D0	00709	MOVL	#1, IN_SCOPE	2081	
	34	AE	2C	AE	D1	0070D	CMPL	SCPTR, SYMSCOPE	2084	
				1E	13	00712	BEQL	100\$		
50	2C	AE		14	C1	00714	ADDL3	#20, SCPTR, R0	2085	
		01		60	91	00719	CMPB	(R0), #1		
				05	12	0071C	BNEQ	99\$		
			58	AE	D4	0071E	CLRL	IN_SCOPE	2088	
				0F	11	00721	BRB	100\$	2087	
50	2C	AE		10	C1	00723	ADDL3	#16, SCPTR, R0	2092	
	2C	AE		60	D0	00728	MOVL	(R0), SCPTR		
			0080	CE	D6	0072C	INCL	DEFDEPTH	2093	
				DB	11	00730	BRB	98\$	2082	
		32	14	AE	E9	00732	BLBC	HAVE_LINE_NUM, 103\$	2101	
		2E	58	AE	E9	00736	BLBC	IN_SCOPE, 103\$		
		2A	10	AE	E8	0073A	BLBS	LINE_NUM_IS_LAST, 103\$		
			58	AE	D4	0073E	CLRL	IN_SCOPE	2104	
	2C	AE	20	AE	D0	00741	MOVL	RSTPTR, SCPTR	2105	
50	2C	AE		14	C1	00746	ADDL3	#20, SCPTR, R0	2106	
		01		60	91	0074B	CMPB	(R0), #1		
				18	13	0074E	BEQL	103\$		
	48	AE	2C	AE	D1	00750	CMPL	SCPTR, LINE_LEX_PTR	2108	
				06	12	00755	BNEQ	102\$		
	58	AE		01	D0	00757	MOVL	#1, IN_SCOPE	2111	
				0B	11	0075B	BRB	103\$	2110	
50	2C	AE		10	C1	0075D	ADDL3	#16, SCPTR, R0	2115	
	2C	AE		60	D0	00762	MOVL	(R0), SCPTR		
				DE	11	00766	BRB	101\$	2106	
		03	58	AE	E8	00768	BLBS	IN_SCOPE, 104\$	2125	
				009F	31	0076C	BRW	108\$		
7E		56		01	78	0076F	ASHL	#1, STKPTR, -(SP)	2132	
		6E		02	C0	00773	ADDL2	#2, (SP)		
	00000000G	00		01	FB	00776	CALLS	#1, DBG\$GET_MEMORY		
		30		50	D0	0077D	MOVL	R0, CANDBLK		
				5A	D4	00781	CLRL	J	2133	
				50	D4	00783	CLRL	I	2134	

				28	11	00785	BRB	106\$		
				00A6	CE40	7F 00787	105\$: PUSHAQ	SYMSTACK+6[I]	2136	
					9E	B5 0078C	TSTW	@(SP)+		
					1F	12 0078E	BNEQ	106\$		
				30	BE4A	7F 00790	PUSHAQ	@CANDBLK[J]	2139	
				00A4	CE40	7F 00794	PUSHAQ	SYMSTACK[I]		
					9E	D0 00799	MOVL	@(SP)+, @(SP)+		
51		30	9E		04	C1 0079C	ADDL3	#4, CANDBLK, R1	2140	
			52		614A	7E 007A1	MOVAQ	(R1)[J], R2		
				00A4	CE40	7F 007A5	PUSHAQ	SYMSTACK+4[I]		
			62		9E	3C 007AA	MOVZWL	@(SP)+, (R2)		
					5A	D6 007AD	INCL	J	2141	
D4			50		56	F3 007AF	106\$: AOBLEQ	STKPTR, I, 105\$	2134	
				30	BE4A	7F 007B3	PUSHAQ	@CANDBLK[J]	2146	
					9E	D4 007B7	CLRL	@(SP)+		
50		30	AE		04	C1 007B9	ADDL3	#4, CANDBLK, R0	2147	
			51		604A	7E 007BE	MOVAQ	(R0)[J], R1		
			61		0080	CE	D0 007C2	MOVL	DEFDEPTH, (R1)	
					58	D6 007C7	INCL	NCANDS	2154	
			50		00000000'	EF	D0 007C9	MOVL	CANDLST, R0	2155
			60			58	D1 007D0	CMPL	NCANDS, (R0)	
					30	15 007D3	BLEQ	107\$		
			60		0A	C0 007D5	ADDL2	#10, (R0)	2158	
		7C	AE		50	D0 007D8	MOVL	R0, OLDCAND	2159	
7E			60		01	C1 007DC	ADDL3	#1, (R0), -(SP)	2160	
		00000000G	00		01	FB 007E0	CALLS	#1, DBG\$GET_MEMORY		
		00000000'	EF		50	D0 007E7	MOVL	R0, CANDLST		
			50		02	78 007EE	ASHL	#2, NCANDS, R0	2161	
00000000'			FF		7C	BE	50	28, 0C7F2	MOV3	R0, @OLDCAND, @CANDLST
				7C	AE	DD 007FB	PUSHL	OLDCAND	2162	
		00000000G	00		01	FB 007FE	CALLS	#1, DBG\$REL_MEMORY		
		00000000'FF	48		30	AE	D0 00805	107\$: MOVL	CANDBLK, @CANDLST[NCANDS]	2165
					56	D5 0080E	108\$: TSTL	STKPTR	2175	
					23	15 00810	BLEQ	110\$		
				00A6	CE46	7F 00812	PUSHAQ	SYMSTACK+6[STKPTR]	2177	
			50		9E	3C 00817	MOVZWL	@(SP)+, R0		
					15	13 0081A	BEQL	109\$		
			44	AE	50	D0 0081C	MOVL	R0, TPINDEX	2180	
					00A0	CE46	7F 00820	PUSHAQ	SYMSTACK[STKPTR]	2181
			59		9E	D0 00825	MOVL	@(SP)+, RPTR		
44	AE	1A	A9		00	ED 00828	CMPIV	#0, #16, 26(RPTR), TPINDEX	2182	
					04	14 0082F	BGTR	110\$		
					56	D7 00831	109\$: DECL	STKPTR	2185	
					D9	11 00833	BRB	108\$	2175	
					56	D5 00835	110\$: TSTL	STKPTR	2188	
					4B	13 00837	BEQL	113\$		
				00A6	CE46	7F 00839	PUSHAQ	SYMSTACK+6[STKPTR]	2196	
			9E		01	A1 0083E	ADDW3	#1, TPINDEX, @(SP)+		
			48	AE	1C	A9	D0 00843	MOVL	28(RPTR), TPTR	2197
			50		44	AE	D0 00848	MOVL	TPINDEX, R0	2198
			59		68	BE40	D0 0084C	MOVL	@TPTR[R0], RPTR	
				00A4	CE46	7F 00851	PUSHAQ	SYMSTACK+4[STKPTR]	2199	
			24	AE	9E	3C 00856	MOVZWL	@(SP)+, PINDEX		
					24	AE	D6 0085A	INCL	PIINDEX	
			03		24	AE	F4 0085D	111\$: SOBGEQ	PIINDEX, 112\$	2207
					24	AE	D4 00861	CLRL	PIINDEX	2208
			01		14	A9	91 00864	112\$: CMPB	20(RPTR), #1	2217

```

M 6
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACKACCESS.B32:1

```

RV

[illegible]

			52	DD	0091A	125\$:	PUSHL	R2	2290
			58	DD	0091C		PUSHL	NCANDS	
		04	AC	DD	0091E		PUSHL	PATHNAME	
	0000V	CF	03	FB	00921		CALLS	#3, SCOPE_RULE_PLI	
	4C	AE	50	DO	00926	126\$:	MOVL	R0, GOOD_CAND	
			0F	15	0092A		BLEQ	127\$	2299
		50	4C	AE	DO	0092C	MOVL	GOOD_CAND, R0	2302
	30	AE	00000000	'FF	40	DO	00930	MOVL	@CANDLIST[R0], CANDBLK
			2D	11	00939		BRB	131\$	2301
			52	D4	0093B	127\$:	CLRL	I	2310
			0E	11	0093D		BRB	129\$	
		00000000	'FF	42	DD	0093F	128\$:	PUSHL	@CANDLIST[I]
EE	00000000G	00	01	FB	00946		CALLS	#1, DBG\$REL_MEMORY	
		52	58	F3	0094D	129\$:	AOBLEQ	NCANDS, I, T28\$	
	FFFFFFFF	8F	4C	AE	D1	00953	CLRL	NCANDS	2311
			03	13	0095B		CMPL	GOOD_CAND, #-1	2318
			F9C1	31	0095D		BEQL	130\$	
		08	BC	D4	00960	130\$:	BRW	37\$	
	0C	BC	09	DO	00963		CLRL	@SYMID	2321
			04	00967		MOVL	#9, @KIND		2322
			5A	D4	00968	131\$:	RET		2320
		30	BE	4A	7F	0096A	132\$:	CLRL	J
			9E	DO	0096E		PUSHAQ	@CANDBLK[J]	2336
		59	51	13	00971		MOVL	@(SP)+, RPTR	2339
		0A	A9	91	00973		BEQL	133\$	2340
			4B	12	00977		CMPB	20(RPTR), #10	2341
			07	DD	00979		BNEQ	133\$	
	00000000G	00	01	FB	0097B		PUSHL	#7	2342
	20	AE	50	DO	00982		CALLS	#1, DBG\$GET_MEMORY	
	20	BE	00	DO	00986		MOVL	R0, RSTPTR	
	00000000G	00	20	AE	DO	0098E	MOVL	RST\$TEMP_LIST, @RSTPTR	2343
50	20	AE	0C	C1	00996		MOVL	RSTPTR, RST\$TEMP_LIST	2344
		60	A9	DO	0099B		ADDL3	#12, RSTPTR, R0	2345
50	20	AE	10	C1	0099F		MOVL	12(RPTR), (R0)	
		60	A9	DO	009A4		ADDL3	#16, RSTPTR, R0	2346
50	20	AE	14	C1	009A8		MOVL	16(RPTR), (R0)	
			60	94	009AD		ADDL3	#20, RSTPTR, R0	2347
50	20	AE	18	C1	009AF		CLRB	(R0)	
		60	A9	DO	009B4		ADDL3	#24, RSTPTR, R0	2348
		30	BE	4A	7F	009B8	MOVL	24(RPTR), (R0)	
		9E	24	AE	DO	009BC	PUSHAQ	@CANDBLK[J]	2349
			5A	D6	009C0		MOVL	RSTPTR, @(SP)+	
			A6	11	009C2		INCL	J	2350
			5A	D4	009C4	133\$:	BRB	132\$	2337
		30	BE	4A	7F	009C6	134\$:	CLRL	J
	20	AE	9E	DO	009CA		PUSHAQ	@CANDBLK[J]	2357
			2B	13	009CE		MOVL	@(SP)+, RSTPTR	2360
50	20	AE	14	C1	009D0		BEQL	136\$	2361
			60	95	009D5		ADDL3	#20, RSTPTR, R0	2362
			22	12	009D7		(R0)		
50	20	AE	14	C1	009D9		TSTB	136\$	
		60	90	009DE		BNEQ	#20, RSTPTR, R0		2363
51	30	AE	08	C1	009E1		ADDL3	#6, (R0)	
		52	614A	7E	009E6		MOVB	#8, CANDBLK, R1	2364
		50	62	DO	009EA		ADDL3	(R1)[J], R2	
			08	13	009ED		MOVAQ	(R1)[J], R2	
						MOVL	(R2), R0		
						BEQL	135\$		

51	20	AE	10	C1	009EF	ADDL3	#16, RSTPTR, R1	2366
		61	50	D0	009F4	MOVL	R0, (R1)	
			5A	D6	009F7	INCL	J	2368
			CB	11	009F9	BRB	134\$	2358
		53	10	AE	E9	009FB	136\$: BLBC	LINE_NUM_IS_LAST, 139\$
	0090	CE	30	BE	D0	009FF	137\$: MOVL	@CANDBLK, MODRSTPTR
		50	0090	CE	D0	00A05	137\$: MOVL	MODRSTPTR, R0
		01	14	A0	91	00A0A	CMPB	20(R0), #1
				08	13	00A0E	BEQL	138\$
	0090	CE	10	A0	D0	00A10	MOVL	16(R0), MODRSTPTR
				ED	11	00A16	BRB	137\$
				7E	D4	00A18	138\$: CLRL	-(SP)
			0098	CE	9F	00A1A	PUSHAB	LINEEND
			00A0	CE	9F	00A1E	PUSHAB	LINESTART
			009C	CE	DD	00A22	PUSHL	MODRSTPTR
			10	AE	DD	00A26	PUSHL	STMT_NUM
			18	AE	DD	00A29	PUSHL	LINE_NUM
	00000000G	00	06	FB	00A2C	CALLS	#6, DBG\$LINE_TO_PC_LOOKUP	
	0084	CF	50	D0	00A33	MOVL	R0, STATUS	
			0094	CE	DD	00A38	PUSHL	LINEEND
			009C	CE	DD	00A3C	PUSHL	LINESTART
			08	AE	DD	00A40	PUSHL	STMT_NUM
			10	AE	DD	00A43	PUSHL	LINE_NUM
			40	BE	DD	00A46	PUSHL	@CANDBLK
	0000V	CF	05	FB	00A49	CALLS	#5, DBG\$STA_LINE_NUM_RST	
	30	BE	50	D0	00A4E	MOVL	R0, @CANDBLK	
	20	AE	30	BE	D0	00A52	139\$: MOVL	@CANDBLK, RSTPTR
00	04	BC	10	ED	00A57	CMPZV	#16, #8, @PATHNAME, #0	2391
			03	12	00A5D	BNEQ	140\$	2398
			00A5	31	00A5F	BRW	146\$	
		03	60	AE	E9	00A62	140\$: BLBC	HAVE_NUM_SCOPE, 141\$
				00A2	31	00A66	BRW	147\$
		52	20	AE	D0	00A69	141\$: MOVL	RSTPTR, ROUTPTR
		02	14	A2	91	00A6D	142\$: CMPB	20(ROUTPTR), #2
				2D	13	00A71	BEQL	144\$
		01	14	A2	91	00A73	CMPB	20(ROUTPTR), #1
				21	12	00A77	BNEQ	143\$
			009C	CE	9F	00A79	PUSHAB	PATHSTRING
			04	AC	DD	00A7D	PUSHL	PATHNAME
	00000000G	00	02	FB	00A80	CALLS	#2, DBG\$NPATHTDESC_TO_CS	
			009C	CE	DD	00A87	PUSHL	PATHSTRING
			01	DD	00A8B	PUSHL	#1	
			00028C90	8F	DD	00A8D	PUSHL	#167056
	00000000G	00	03	FB	00A93	CALLS	#3, LIB\$SIGNAL	
		52	10	A2	D0	00A9A	143\$: MOVL	16(ROUTPTR), ROUTPTR
				CD	11	00A9E	BRB	142\$
50	04	BC	10	EF	00AA0	144\$: EXTZV	#16, #8, @PATHNAME, R0	2406
		08	FC	A740	D0	00AA6	MOVL	-4(PATHVEC)[R0], PNAME
		AE	0C	A2	DD	00AAC	PUSHL	12(ROUTPTR)
	00000000G	00	01	FB	00AAF	CALLS	#1, DBG\$GET_DST_NAME	2407
	3C	AE	50	D0	00AB6	MOVL	R0, RNAME	2423
		51	08	BE	9A	00ABA	MOVZBL	@PNAME, R1
		50	3C	BE	9A	00ABE	MOVZBL	@RNAME, R0
	54	3C	01	C1	00AC2	ADDL3	#1, RNAME, R4	2424
	55	08	01	C1	00AC7	ADDL3	#1, PNAME, R5	2425
50	00	65	51	2D	00ACC	CMPCS	R1, (R5), #0, R0, (R4)	
			64		00AD1			

				009C	21	13	00AD2	BEQL	145\$	
				04	CE	9F	00AD4	PUSHAB	PATHSTRING	2428
					AC	DD	00AD8	PUSHL	PATHNAME	
		00000000G	00		02	FB	00ADB	CALLS	#2, DBG\$NPATHDESC_TO_CS	
				009C	CE	DD	00AE2	PUSHL	PATHSTRING	2429
					01	DD	00AE6	PUSHL	#1	
				00C28C90	8F	DD	00AE8	PUSHL	#167056	
		00000000G	00		03	FB	00AEE	CALLS	#3, LIB\$SIGNAL	
50		04	AC		04	C1	00AF5	145\$: ADDL3	#4, PATHNAME, R0	2436
					60	D5	00AFA	TSTL	(R0)	
					41	13	00AFC	BEQL	151\$	
52		04	AC		04	C1	00AFE	ADDL3	#4, PATHNAME, R2	2438
					62	DD	00B03	PUSHL	(R2)	
			14		0E	11	00B05	BRB	148\$	
				60	AE	E9	00B07	146\$: BLBC	HAVE_NUM_SCOPE, 149\$	2447
				0088	CE	D5	00B08	147\$: TSTL	NUMSCP_INVOC_NUM	
					0E	13	00B0F	BEQL	149\$	
				0088	CE	DD	00B11	PUSHL	NUMSCP_INVOC_NUM	2449
				24	AE	DD	00B15	148\$: PUSHL	RSTPTR	
		F201	CF		02	FB	00B18	CALLS	#2, DBG\$BUILD_INVOC_RST	
00	04	BC	08		1C	11	00B1D	BRB	150\$	
					10	ED	00B1F	149\$: CMPZV	#16, #8, @PATHNAME, #0	2456
01	04	BC	08		18	12	00B25	BNEQ	151\$	
					08	ED	00B27	CMPZV	#8, #8, @PATHNAME, #1	2457
					10	12	00B2D	BNEQ	151\$	
				008C	CE	D5	00B2F	PUSHL	SCOPE	2459
				24	AE	DD	00B33	PUSHL	RSTPTR	
		0000V	CF		02	FB	00B36	CALLS	#2, FOLLOW_STATIC_LINK	
		20	AE		50	DD	00B3B	150\$: MOVL	R0, RSTPTR	
		08	BC	20	AE	DD	00B3F	151\$: MOVL	RSTPTR, @SYMID	2464
50		20	AE		14	C1	00B44	ADDL3	#20, RSTPTR, R0	2465
		0C	BC		60	9A	00B49	MOVZBL	(R0), @KIND	
					52	D4	00B4D	CLRL	I	2470
					0E	11	00B4F	BRB	153\$	
				00000000'FF	42	DD	00B51	152\$: PUSHL	@CANDLIST[I]	
EE		00000000G	00		01	FB	00B58	CALLS	#1, DBG\$REL_MEMORY	
			52		58	F3	00B5F	153\$: AOBLEQ	NCANDS, I, T52\$	
			50	00000000G	00	DD	00B63	MOVL	RST\$REF_LIST, R0	2478
			60	04	A0	D1	00B6A	CMPL	4(R0), (R0)	
					39	12	00B6E	BNEQ	154\$	
			60		14	C0	00B70	ADDL2	#20, (R0)	2481
7E			60		02	C1	00B73	ADDL3	#2, (R0), -(SP)	2482
		00000000G	00		01	FB	00B77	CALLS	#1, DBG\$GET_MEMORY	
		70	AE		50	DD	00B7E	MOVL	R0, NEWREFLIST	
			57	00000000G	00	DD	00B82	MOVL	RST\$REF_LIST, R7	2483
			50	04	A7	DD	00B89	MOVL	4(R7), R0	
			50		04	C4	00B8D	MULL2	#4, R0	
			50		08	C0	00B90	ADDL2	#8, R0	
70	BE		67		50	28	00B93	MOVC3	R0, (R7), @NEWREFLIST	
					57	DD	00B98	PUSHL	R7	2484
		00000000G	00		01	FB	00B9A	CALLS	#1, DBG\$REL_MEMORY	
		00000000G	00	70	AE	DD	00BA1	MOVL	NEWREFLIST, RST\$REF_LIST	2485
			50	00000000G	00	DD	00BA9	154\$: MOVL	RST\$REF_LIST, R0	2488
				04	A0	D6	00BB0	INCL	4(R0)	
			51	04	A0	DD	00BB3	MOVL	4(R0), R1	2489
		04 A041	20		AE	DD	00BB7	MOVL	RSTPTR, 4(R0)[R1]	
		00000000G	00	0090	CE	D1	00BBD	CMPL	MODRSTPTR, LRUM\$MOST_RECENT	2495

RSTACCESS
V04-000

L 6
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 68
(12)

00000000G 00 0090 0B 13 00BC6 BEQL 155\$
CE DD 00BC8 PUSHL MODRSTPTR
01 FB 00BCC CALLS #1, DBG\$RST_MOST_RECENT
04 00BD3 155\$: RET

:
:
:
: 2498

; Routine Size: 3028 bytes, Routine Base: DBG\$CODE + 03BB

```
2378 2499 1 GLOBAL ROUTINE DBG$STA_GETSYMOFF(ADDR, P_SYMID, P_BIT_OFFSET) =
2379 2500 1
2380 2501 1 FUNCTION:
2381 2502 1 This routine accepts a address descriptor, and attempts to symbolize
2382 2503 1 it as a symbol name plus offset.
2383 2504 1 It always returns the best possible symbolization; if the address can
2384 2505 1 be symbolized by more than one symbol name with the same offset, then
2385 2506 1 the first is chosen to be the best.
2386 2507 1
2387 2508 1 The routine accepts an optional print flag (the default being no print).
2388 2509 1 The best symbolization is returned in the form of symid and offset. If
2389 2510 1 output is specified (as in the SYMBOLIZE command), then the following
2390 2511 1 occurs:
2391 2512 1
2392 2513 1 1. If the address is found to be an instruction address, then the
2393 2514 1 the routines it calls symbolize it as either a label (exact match
2394 2515 1 only), or as a line number and byte offset from the start of the
2395 2516 1 line. This information is printed, along with the routine name plus
2396 2517 1 byte offset from the beginning of the routine.
2397 2518 1
2398 2519 1 2. If the address turns out to be a data address (that is, if it turns out
2399 2520 1 not to be in any routine's instruction address range), this routine will
2400 2521 1 see if it corresponds to any static data item. If so, that data symbol
2401 2522 1 and an offset from it will be printed, and the symid and offset will be
2402 2523 1 returned. Symbolization will not be done to array elements or record
2403 2524 1 components--only the outer level static data item will be returned as
2404 2525 1 as the symbol. If the address is a stack address, then the VAX call
2405 2526 1 stack is searched for a match. Or, if the address is a register
2406 2527 1 address, then the module's symbol table is searched for those symbols
2407 2528 1 bound to that register. If no symbol is found at all, then the symid
2408 2529 1 is set to zero, and the absolute virtual address is returned as the
2409 2530 1 offset. A message saying that no symbolization was possible is
2410 2531 1 is displayed, and the routine returns false.
2411 2532 1
2412 2533 1 DBG$STA_GETSYMOFF is called to symbolize addresses only in certain cir-
2413 2534 1 cumstances. One is when the user program has faulted somewhere (with an
2414 2535 1 access violation, for example) and the fault address must be symbolized
2415 2536 1 and displayed in an understandable form. Another is when VAX machine
2416 2537 1 instructions are displayed symbolically (through E/I or STEP, for exam-
2417 2538 1 ple) and operands must be displayed in as symbolic a form as possible.
2418 2539 1 DBG$STA_GETSYMOFF is always called during execution of the SYMBOLIZE
2419 2540 1 command, and output is always done in that case.
2420 2541 1
2421 2542 1 INPUTS:
2422 2543 1 ADDR - The address of an address descriptor (byte and bit offset).
2423 2544 1
2424 2545 1 P_SYMID - The address of a longword location where the "symbol identi-
2425 2546 1 fier" should be returned. The "symbol identifier" is a value
2426 2547 1 which uniquely identifies the returned symbol. This value is
2427 2548 1 not directly understood outside the symbol table access rou-
2428 2549 1 tines, but can be passed to various other symbol table access
2429 2550 1 routines to extract information about the symbol.
2430 2551 1
2431 2552 1 P_BIT_OFFSET - The address of a longword location where the bit offset from
2432 2553 1 the SYMID symbol should be returned.
2433 2554 1
2434 2555 1 An optional print flag may be specified. The default is FALSE - no print.
```

```
2435 2556 1
2436 2557 1
2437 2558 1
2438 2559 1
2439 2560 1
2440 2561 1
2441 2562 1
2442 2563 1
2443 2564 1
2444 2565 1
2445 2566 1
2446 2567 1
2447 2568 1
2448 2569 1
2449 2570 1
2450 2571 1
2451 2572 2
2452 2573 2
2453 2574 2
2454 2575 2
2455 2576 2
2456 2577 2
2457 2578 2
2458 2579 2
2459 2580 2
2460 2581 2
2461 2582 2
2462 2583 2
2463 2584 2
2464 2585 2
2465 2586 2
2466 2587 2
2467 2588 2
2468 2589 2
2469 2590 2
2470 2591 2
2471 2592 2
2472 2593 2
2473 2594 2
2474 2595 2
2475 2596 2
2476 2597 2
2477 2598 2
2478 2599 2
2479 2600 2
2480 2601 2
2481 2602 2
2482 2603 2
2483 2604 2
2484 2605 2
2485 2606 2
2486 2607 2
2487 2608 2
2488 2609 2
2489 2610 2
2490 2611 2
2491 2612 3
```

OUTPUTS:

SYMID - A symbol identifier which uniquely identifies the symbol which best symbolizes ADDR is returned to SYMID. This symbol identifier can then be passed to any symbol table access routine which accepts a SYMID parameter. If no suitable symbol can be found, a zero is returned to SYMID.

OFFSET - The bit offset of ADDR relative to the SYMID symbol is returned to OFFSET. If (SYMID) is zero, this offset is simply the original address descriptor.

The routine returns true if symbolization was possible; otherwise it returns false.

BEGIN

BUILTIN
ACTUALCOUNT,
ACTUALPARAMETER;

BIND
SYMID = .P_SYMID: REF RST\$ENTRY,
BIT_OFFSET = .P_BIT_OFFSET;

MAP
ADDR: REF DBG\$ADDRESS_DESC; ! Pointer to address descriptor

LOCAL
PRINT_FLAG; ! Flag for print/no print.

! If the caller wants output, then the fourth parameter will be true.
! Otherwise, the fourth parameter will be false, or not at all.

IF ACTUALCOUNT() GEQ 4
THEN
PRINT_FLAG = ACTUALPARAMETER (4)
ELSE
PRINT_FLAG = FALSE;

! See if the address is a register address.
IF DBG\$SYMBOLIZE_REG (.ADDR, SYMID, BIT_OFFSET, .PRINT_FLAG)
THEN
RETURN TRUE;

! See if the address is a static address.
IF DBG\$SEARCH_SAT (.ADDR, SYMID, BIT_OFFSET, .PRINT_FLAG)
THEN
BEGIN

```
: 2492 2613 3      ! At this point, we have found a module, but there is no symid we could
: 2493 2614 3      ! locate, so try global search to locate more info.
: 2494 2615 3
: 2495 2616 3      IF .SYMID EQL 0
: 2496 2617 3      THEN
: 2497 2618 4          BEGIN
: 2498 2619 4              IF .PRINT_FLAG THEN DBG$PRINT CONTROL(DBG$K_PRT_RESET);
: 2499 2620 4              IF DBG$SEARCH_GLOBAL (.ADDR, SYMID, BIT_OFFSET, .PRINT_FLAG)
: 2500 2621 4                  THEN
: 2501 2622 4                  RETURN TRUE
: 2502 2623 4
: 2503 2624 4              ELSE
: 2504 2625 5                  BEGIN
: 2505 2626 5                      BIT_OFFSET = .ADDR;
: 2506 2627 5                      RETURN FALSE;
: 2507 2628 4                  END;
: 2508 2629 4
: 2509 2630 4              END
: 2510 2631 4
: 2511 2632 4      ! We have found a good symid in a given module, for SYMBOLIZE command,
: 2512 2633 4      ! we print a bit more info for the symbol declared as global. But we
: 2513 2634 4      ! do not want the global symid.
: 2514 2635 4
: 2515 2636 4      ELSE
: 2516 2637 3          BEGIN
: 2517 2638 4              IF .PRINT_FLAG
: 2518 2639 4                  THEN
: 2519 2640 4                  BEGIN
: 2520 2641 5                      LOCAL
: 2521 2642 5                      TMP_SYMID, TMP_OFFSET;
: 2522 2643 5
: 2523 2644 5                      DBG$PRINT CONTROL(DBG$K_PRT_RESET);
: 2524 2645 5                      DBG$SEARCH_GLOBAL (.ADDR, TMP_SYMID, TMP_OFFSET, .PRINT_FLAG);
: 2525 2646 5                      END;
: 2526 2647 4
: 2527 2648 4                  RETURN TRUE;
: 2528 2649 4
: 2529 2650 3                  END;
: 2530 2651 3
: 2531 2652 2      END;
: 2532 2653 2
: 2533 2654 2      ! See if the address is on the call stack.
: 2534 2655 2
: 2535 2656 2      IF DBG$SEARCH_VAX_CALL_STACK (.ADDR, SYMID, BIT_OFFSET, .PRINT_FLAG)
: 2536 2657 2      THEN
: 2537 2658 2          RETURN TRUE;
: 2538 2659 2
: 2539 2660 2
: 2540 2661 2      ! Try once more!!! See if the address is a global symbol.
: 2541 2662 2
: 2542 2663 2      IF DBG$SEARCH_GLOBAL (.ADDR, SYMID, BIT_OFFSET, .PRINT_FLAG)
: 2543 2664 2      THEN
: 2544 2665 2          RETURN TRUE;
: 2545 2666 2
: 2546 2667 2
: 2547 2668 2
: 2548 2669 2      ! The address was not found, and symbolization was thus impossible.
```

```
: 2549      2670  2      !  
: 2550      2671  2      SYMID = 0;  
: 2551      2672  2      BIT_OFFSET = .ADDR;  
: 2552      2673  2      RETURN FALSE;  
: 2553      2674  1      END;
```

			00FC 00000	.ENTRY	DBG\$STA GETSYMOFF, Save R2,R3,R4,R5,R6,R7	2499
57	00000000G	00	9E 00002	MOVAB	DBG\$PRINT_CONTROL, R7	
56	00000000G	00	9E 00009	MOVAB	DBG\$SEARCH_GLOBAL, R6	
5E		08	C2 00010	SUBL2	#8, SP	
54	08	AC	D0 00013	MOVL	P_SYMID, R4	2579
53	0C	AC	D0 00017	MOVL	P_BIT_OFFSET, R3	2580
04		6C	91 0001B	CMPB	(AP), #4	2592
		06	1F 0001E	BLSSU	1\$	
55	10	AC	D0 00020	MOVL	16(AP), PRINT_FLAG	2594
		02	11 00024	BRB	2\$	
		55	D4 00026	CLRL	PRINT_FLAG	2596
		28	BB 00028	PUSHR	#*M<R3,R5>	2601
		54	DD 0002A	PUSHL	R4	
52	04	AC	D0 0002C	MOVL	ADDR, R2	
		52	DD 00030	PUSHL	R2	
00000000G	00	04	FB 00032	CALLS	#4, DBG\$SYMBOLIZE_REG	
55		50	E8 00039	BLBS	R0, 6\$	
		28	BB 0003C	PUSHR	#*M<R3,R5>	2608
		14	BB 0003E	PUSHR	#*M<R2,R4>	
00000000G	00	04	FB 00040	CALLS	#4, DBG\$SEARCH_SAT	
2F		50	E9 00047	BLBC	R0, 5\$	
		64	D5 0004A	TSTL	(R4)	2616
		14	12 0004C	BNEQ	4\$	
05		55	E9 0004E	BLBC	PRINT_FLAG, 3\$	2619
		05	DD 00051	PUSHL	#5	
67		01	FB 00053	CALLS	#1, DBG\$PRINT_CONTROL	
		28	BB 00056	PUSHR	#*M<R3,R5>	2620
		14	BB 00058	PUSHR	#*M<R2,R4>	
66		04	FB 0005A	CALLS	#4, DBG\$SEARCH_GLOBAL	
31		50	E8 0005D	BLBS	R0, 6\$	
		35	11 00060	BRB	8\$	2626
2C		55	E9 00062	BLBC	PRINT_FLAG, 6\$	2639
		05	DD 00065	PUSHL	#5	2645
67		01	FB 00067	CALLS	#1, DBG\$PRINT_CONTROL	
		55	DD 0006A	PUSHL	PRINT_FLAG	2646
	04	AE	9F 0006C	PUSHAB	TMP_OFFSET	
	0C	AE	9F 0006F	PUSHAB	TMP_SYMID	
		52	DD 00072	PUSHL	R2	
66		04	FB 00074	CALLS	#4, DBG\$SEARCH_GLOBAL	
		18	11 00077	BRB	6\$	2649
		28	BB 00079	PUSHR	#*M<R3,R5>	2657
		14	BB 0007B	PUSHR	#*M<R2,R4>	
00000000G	00	04	FB 0007D	CALLS	#4, DBG\$SEARCH_VAX_CALL_STACK	
0A		50	E8 00084	BLBS	R0, 6\$	
		28	BB 00087	PUSHR	#*M<R3,R5>	2664
		14	BB 00089	PUSHR	#*M<R2,R4>	
66		04	FB 0008B	CALLS	#4, DBG\$SEARCH_GLOBAL	

RSTACK
V04-000

D 7
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACK.B32;1

Page 73
(13)

04	50	E9 0008E	BLBC	R0, 7\$:
50	01	D0 00091 6\$:	MOVL	#1, R0	: 2666
		04 00094	RET		:
	64	D4 00095 7\$:	CLRL	(R4)	: 2671
63	52	D0 00097 8\$:	MOVL	R2, (R3)	: 2672
	50	D4 0009A	CLRL	R0	: 2674
		04 0009C	RET		:

; Routine Size: 157 bytes, Routine Base: DBG\$CODE + 0F8F

```
2555 2675 1 GLOBAL ROUTINE DBG$STA_LINE_NUM_RST(LEXPTR, LINE_NUM, STMT_NUM, LINESTART, LINEEND) =
2556 2676 1
2557 2677 1 FUNCTION
2558 2678 1 This routine builds a Line Number RST Entry for a specified line and
2559 2679 1 links it into the RST. In addition to the RST entry, a dummy Label DST
2560 2680 1 Record is built (in the same memory block) to contain the line's name
2561 2681 1 as Counted ASCII (e.g. '%LINE 25.3'). The RST entry is linked into the
2562 2682 1 Temporary RST Entry List pointed to by RST$TEMP_LIST.
2563 2683 1
2564 2684 1 INPUTS
2565 2685 1 LEXPTR - A pointer to the lexical entity RST entry to which the Line
2566 2686 1 Number RST Entry should be attached via the up-scope pointer.
2567 2687 1
2568 2688 1 LINE_NUM - The line's line number.
2569 2689 1
2570 2690 1 STMT_NUM - The line's statement number.
2571 2691 1
2572 2692 1 LINESTART - The line's start address in virtual memory.
2573 2693 1
2574 2694 1 LINEEND - The line's end address in virtual memory.
2575 2695 1
2576 2696 1 OUTPUTS
2577 2697 1 A pointer to the line's Line Number RST Entry is returned as the
2578 2698 1 routine's value.
2579 2699 1
2580 2700 1 BEGIN
2581 2701 2
2582 2702 2 LOCAL
2583 2703 2
2584 2704 2 DSTPTR: REF DST$RECORD,      ! Pointer to dummy Label DST Record
2585 2705 2 J,                          ! Index into the TEXT array; number of
2586 2706 2                               characters in the line's name
2587 2707 2 NAMEPTR: REF VECTOR[.BYTE],  ! Pointer to DST record name vector
2588 2708 2 NUMBER,                      ! Used to convert the statement and line
2589 2709 2                               numbers to ASCII
2590 2710 2 RSTPTR: REF RST$ENTRY,      ! Pointer to Line Number RST Entry
2591 2711 2 TEXT: VECTOR[40,.BYTE];     ! Vector used to generate ASCII name
2592 2712 2
2593 2713 2
2594 2714 2
2595 2715 2 ! Build the line number name as an ASCII string in the TEXT vector. Note
2596 2716 2 ! that the string is stored backward in this vector since we generate low-
2597 2717 2 ! order numeric digits before high-order ones.
2598 2718 2
2599 2719 2 J = 0;
2600 2720 2 IF .STMT_NUM NEQ 0
2601 2721 2 THEN
2602 2722 3 BEGIN
2603 2723 3 NUMBER = .STMT_NUM;
2604 2724 3 WHILE TRUE DO
2605 2725 4 BEGIN
2606 2726 4 TEXT[J] = (.NUMBER MOD 10) + '0';
2607 2727 4 J = J + 1;
2608 2728 4 NUMBER = .NUMBER/10;
2609 2729 4 IF .NUMBER EQL 0 THEN EXITLOOP;
2610 2730 3 END;
2611 2731 3
```



```
2612 2732 3      TEXT[.J] = '.';
2613 2733 3      J = .J + 1;
2614 2734 2      END;
2615 2735 2
2616 2736 2      NUMBER = .LINE_NUM;
2617 2737 2      WHILE TRUE DO
2618 2738 3          BEGIN
2619 2739 3              TEXT[.J] = (.NUMBER MOD 10) + '0';
2620 2740 3              J = .J + 1;
2621 2741 3              NUMBER = .NUMBER/10;
2622 2742 3              IF .NUMBER EQL 0 THEN EXITLOOP;
2623 2743 2          END;
2624 2744 2
2625 2745 2      J = .J + 6;
2626 2746 2      TEXT[.J - 1] = 'X';
2627 2747 2      TEXT[.J - 2] = 'L';
2628 2748 2      TEXT[.J - 3] = 'I';
2629 2749 2      TEXT[.J - 4] = 'N';
2630 2750 2      TEXT[.J - 5] = 'E';
2631 2751 2      TEXT[.J - 6] = '.';
2632 2752 2
2633 2753 2      ! Allocate enough space for the Line Number RST Entry and for a Label DST
2634 2754 2      ! record (which we will build in the same memory block).
2635 2755 2
2636 2756 2      RSTPTR = DBG$GET_MEMORY(RST$K_LINENTSIZE + (.J + 1)/4);
2637 2757 2      DSTPTR = .RSTPTR + 4*RST$K_LINENTSIZE;
2638 2758 2
2639 2759 2
2640 2760 2      ! Construct the dummy Label DST record for the line number.
2641 2761 2
2642 2762 2      DSTPTR[DST$B_LENGTH] = 7 + .J;
2643 2763 2      DSTPTR[DST$B_TYPE] = DST$K_LABEL;
2644 2764 2      DSTPTR[DST$B_VFLAGS] = 0;
2645 2765 2      DSTPTR[DST$L_VALUE] = .LINESTART;
2646 2766 2      NAMEPTR = DSTPTR[DST$B_NAME];
2647 2767 2      NAMEPTR[0] = .J;
2648 2768 2      INCR I FROM 1 TO .J DO NAMEPTR[.I] = .TEXT[.J - .I];
2649 2769 2
2650 2770 2
2651 2771 2
2652 2772 2      ! Then construct the Line Number RST Entry for the line.
2653 2773 2
2654 2774 2      RSTPTR[RST$L_DSTPTR] = .DSTPTR;
2655 2775 2      RSTPTR[RST$L_UPSCOPEPTR] = .LEXPTR;
2656 2776 2      RSTPTR[RST$B_KIND] = RST$K_LINE;
2657 2777 2      RSTPTR[RST$L_STARTADDR] = .LINESTART;
2658 2778 2      RSTPTR[RST$L_ENDADDR] = .LINEEND;
2659 2779 2
2660 2780 2
2661 2781 2      ! Link the RST entry into the Temporary RST Entry List.
2662 2782 2
2663 2783 2      RSTPTR[RST$L_HASH_FLINK] = .RST$TEMP_LIST;
2664 2784 2      RST$TEMP_LIST = .RSTPTR;
2665 2785 2
2666 2786 2
2667 2787 2      ! Return to the caller with the RST entry address as the routine value.
2668 2788 2
```

```
: 2669      2789 2  RETURN .RSTPTR;  
: 2670      2790 2  
: 2671      2791 1  END;
```

				007C 00000	.ENTRY	DBG\$STA LINE NUM RST, Save R2,R3,R4,R5,R6	: 2675
		56	00000000G	00 9E 00002	MOVAB	RST\$TEMP_LIST, R8	
		5E		28 C2 00009	SUBL2	#40, SP	
				52 D4 0000C	CLRL	J	: 2719
			0C	AC D5 0000E	TSTL	STMT_NUM	: 2720
				1C 13 00011	BEQL	2\$	
		51	0C	AC D0 00013	MOVL	STMT_NUM, NUMBER	: 2723
7E	00	51		01 7A 00017	EMUL	#1, NUMBER, #0, -(SP)	: 2726
50	50	8E		0A 7B 0001C	EDIV	#10, (SP)+, R0, R0	
	824E	50		30 81 00021	ADDB3	#48, R0, TEXT[SP]	
		51		0A C6 00026	DIVL2	#10, NUMBER	: 2728
				EC 12 00029	BNEQ	1\$: 2729
		824E		2E 90 0002B	MOVB	#46, TEXT[SP]	: 2732
		51	08	AC D0 0002F	MOVL	LINE_NUM, NUMBER	: 2736
7E	00	51		01 7A 00033	EMUL	#1, NUMBER, #0, -(SP)	: 2739
50	50	8E		0A 7B 00038	EDIV	#10, (SP)+, R0, R0	
	824E	50		30 81 0003D	ADDB3	#48, R0, TEXT[SP]	
		51		0A C6 00042	DIVL2	#10, NUMBER	: 2741
				EC 12 00045	BNEQ	3\$: 2742
		52		05 C0 00047	ADDL2	#5, J	: 2745
		824E		25 90 0004A	MOVB	#37, TEXT-1[SP]	: 2746
	FE AE42	4C		8F 90 0004E	MOVB	#76, TEXT-2[J]	: 2747
	FD AE42	49		8F 90 00054	MOVB	#73, TEXT-3[J]	: 2748
	FC AE42	4E		8F 90 0005A	MOVB	#78, TEXT-4[J]	: 2749
	FB AE42	45		8F 90 00060	MOVB	#69, TEXT-5[J]	: 2750
	FA AE42	20		90 00066	MOVB	#32, TEXT-6[J]	: 2751
		51	0B	A2 9E 0006B	MOVAB	11(R2), R1	: 2757
		51		04 C6 0006F	DIVL2	#4, R1	
			08	A1 9F 00072	PUSHAB	8(R1)	
		00000000G	00	01 FB 00075	CALLS	#1, DBG\$GET MEMORY	
			53	A0 9E 0007C	MOVAB	32(R0), DSTPTR	: 2758
63		52	20	07 81 00080	ADDB3	#7, J, (DSTPTR)	: 2763
	01 A3	8B		8F 9B 00084	MOVZBW	#187, 1(DSTPTR)	: 2764
	03 A3	10		AC D0 00089	MOVL	LINE\$START, 3(DSTPTR)	: 2766
		54	07	A3 9E 0008E	MOVAB	7(R3), NAMEPTR	: 2767
		64		52 90 00092	MOVB	J, (NAMEPTR)	: 2768
				55 D4 00095	CLRL	I	: 2769
				09 11 00097	BRB	5\$	
51		52		55 C3 00099	SUBL3	I, J, R1	
		6544	6E41	90 0009D	MOVB	TEXT[R1], (1)[NAMEPTR]	
F3		55		52 F3 000A2	AOBLEQ	J, I, 4\$	
	0C A0	53		D0 000A6	MOVL	DSTPTR, 12(RSTPTR)	: 2774
	10 A0	04		AC D0 000AA	MOVL	LEXPTR, 16(RSTPTR)	: 2775
	14 A0	05		90 000AF	MOVB	#5, 20(RSTPTR)	: 2776
	18 A0	10		AC 7D 000B3	MOVQ	LINE\$START, 24(RSTPTR)	: 2777
		60		66 D0 000B8	MOVL	RST\$TEMP_LIST, (RSTPTR)	: 2783
		66		50 D0 000BB	MOVL	RSTPTR, RST\$TEMP_LIST	: 2784
				04 000BE	RET		: 2791

RSTACCESS
V04-000

M 7
16-Sep-1984 02:48:17
14-Sep-1984 12:18:26

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]RSTACCESS.B32;1

Page 77
(14)

; Routine Size: 191 bytes, Routine Base: DBG\$CODE + 102C

; 2672 2792 1

R
V(

```
: 2674 2793 1 GLOBAL ROUTINE DBG$STA_LOCK_SYMID(SYMID_LIST_PTR): NOVALUE =
: 2675 2794 1
: 2676 2795 1 FUNCTION
: 2677 2796 1 This routine "locks" a list of SYMIDs in the RST so that the correspond-
: 2678 2797 1 ing RST entries cannot be released to the free memory pool. SYMIDs are
: 2679 2798 1 locked this way only when they will be saved in a Primary Descriptor or
: 2680 2799 1 elsewhere across Debug commands. SYMIDs used to represent "." (current
: 2681 2800 1 location) or breakpoint locations are examples of SYMIDs which must be
: 2682 2801 1 locked across commands. A locked SYMID remains locked until it is ex-
: 2683 2802 1 plicitly unlocked by a call to DBG$STA_UNLOCK_SYMID.
: 2684 2803 1
: 2685 2804 1 The actual locking procedure involves incrementing the Reference Count
: 2686 2805 1 in the SYMID's RST entry and in most RST entries directly accessible
: 2687 2806 1 from this RST entry. This includes all RST entries upscope from the
: 2688 2807 1 present entry and all Data Type RST Entries attached to the up-scope
: 2689 2808 1 chain.
: 2690 2809 1
: 2691 2810 1 INPUTS
: 2692 2811 1 SYMID_LIST_PTR - A pointer to a linked list of Linked List Nodes, where
: 2693 2812 1 each node contains a forward link and a SYMID value. Each
: 2694 2813 1 SYMID on the list is "locked" in the RST by incrementing the
: 2695 2814 1 reference count of the corresponding RST entry.
: 2696 2815 1
: 2697 2816 1 OUTPUTS
: 2698 2817 1 NONE
: 2699 2818 1
: 2700 2819 1
: 2701 2820 2 BEGIN
: 2702 2821 2
: 2703 2822 2 LOCAL
: 2704 2823 2 LISTPTR: REF DBG$LINK_NODE; ! Pointer to current linked list node
: 2705 2824 2
: 2706 2825 2
: 2707 2826 2
: 2708 2827 2 ! Loop through all the SYMIDs (i.e., RST pointers) on the linked list.
: 2709 2828 2 ! For each SYMID on the list, call ADD_TO_REF_COUNT to increment the RST
: 2710 2829 2 ! entry's reference count.
: 2711 2830 2
: 2712 2831 2 LISTPTR = .SYMID_LIST_PTR;
: 2713 2832 2 WHILE .LISTPTR NEQ 0 DO
: 2714 2833 3 BEGIN
: 2715 2834 3 ADD TO REF_COUNT(.LISTPTR[DBG$LINK_NODE_VALUE], +1);
: 2716 2835 3 LISTPTR = .LISTPTR[DBG$LINK_NODE_LINK];
: 2717 2836 2 END;
: 2718 2837 2
: 2719 2838 2 RETURN;
: 2720 2839 2
: 2721 2840 1 END;
```

```
52      04      0004 00000      .ENTRY  DBG$STA_LOCK_SYMID, Save R2
          AC  D0 00002      MOVL     SYMID_LIST_PTR, LISTPTR
          OF  13 00006 1$:    BEQL     2$
          01  D0 00008      PUSHL    #1
```

```
: 2793
: 2831
: 2832
: 2834
```

RSTACK
V04-000

J 7
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACK.B32;1

Page 79
(15)

0000V	CF	04	A2	DD	0000A	PUSHL	4(LISTPTR)
	52		02	FB	0000D	CALLS	#2, ADD TO REF COUNT
			62	D0	00012	MOVL	(LISTPTR), -LISTPTR
			EF	11	00015	BRB	1\$
			04	00017	2\$:	RET	

:
:
: 2835
: 2832
: 2840

; Routine Size: 24 bytes, Routine Base: DBG\$CODE + 10EB

```
2723 2841 1 GLOBAL ROUTINE DBG$STA_LOOKUP_GBL(NAMEPTR) =
2724 2842 1
2725 2843 1 FUNCTION
2726 2844 1 This routine looks up a symbol in the Global Symbol Table (the GST)
2727 2845 1 and only in the GST. It accepts the symbol name as input, looks up
2728 2846 1 that symbol in the GST, and returns a pointer to an RST entry for the
2729 2847 1 global symbol. If the symbol is not found in the GST, a value of zero
2730 2848 1 is returned.
2731 2849 1
2732 2850 1 The whole RST and GST search is suppressed if the DBG$GB_NO_GLOBALS
2733 2851 1 flag is set. In this case, the routine always returns zero.
2734 2852 1
2735 2853 1 INPUTS
2736 2854 1 NAMEPTR - A pointer to the symbol name to be looked up in the GST.
2737 2855 1 The name must be represented by a Counted ASCII string.
2738 2856 1
2739 2857 1 OUTPUTS
2740 2858 1 A pointer to an RST entry for the global symbol is returned as the
2741 2859 1 routine value. If the symbol is not in the GST, zero is
2742 2860 1 returned as the routine value.
2743 2861 1
2744 2862 1
2745 2863 2 BEGIN
2746 2864 2
2747 2865 2 MAP
2748 2866 2 NAMEPTR: REF VECTOR[.BYTE]; ! Pointer to Counted ASCII symbol name
2749 2867 2
2750 2868 2
2751 2869 2 LOCAL
2752 2870 2 RSTPTR: REF RST$ENTRY; ! Pointer to current symbol's RST entry
2753 2871 2 ! symbol is a routine entry point
2754 2872 2
2755 2873 2
2756 2874 2
2757 2875 2 ! If the Global Symbol Table is suppressed, return zero right away.
2758 2876 2
2759 2877 2 IF .DBG$GB_NO_GLOBALS THEN RETURN 0;
2760 2878 2
2761 2879 2
2762 2880 2 ! Search the RST Hash Table for a symbol with the desired name which is
2763 2881 2 ! also marked as being global (meaning that it is derived from the GST).
2764 2882 2 ! If we find such an RST entry, we return its address to the caller.
2765 2883 2
2766 2884 2 DBG$HASH_FIND_SETUP(.NAMEPTR);
2767 2885 2 WHILE TRUE DO
2768 2886 3 BEGIN
2769 2887 3 RSTPTR = DBG$HASH_FIND(.NAMEPTR);
2770 2888 3 IF .RSTPTR EQL 0 THEN EXITLOOP;
2771 2889 3 IF .RSTPTR[RST$V_GLOBAL] THEN RETURN .RSTPTR;
2772 2890 2 END;
2773 2891 2
2774 2892 2
2775 2893 2 ! We did not find the symbol in the Global Symbol Table--just return zero.
2776 2894 2
2777 2895 2 RETURN 0;
2778 2896 2
2779 2897 1 END;
```

			0000	00000		.ENTRY	DBG\$STA_LOOKUP_GBL, Save nothing	:	2841
	1D	00000000G	00	E8	00002	BLBS	DBG\$GB_NO_GLOBALS, 2\$:	2877
			04	AC	DD	PUSHL	NAMEPTR	:	2884
00000000G	00		01	FB	0000C	CALLS	#1, DBG\$HASH_FIND_SETUP	:	
			04	AC	DD	PUSHL	NAMEPTR	:	2887
00000000G	00		01	FB	00016	CALLS	#1, DBG\$HASH_FIND	:	
			50	D5	0001D	TSTL	RSTPTR	:	2888
			05	13	0001F	BEQL	2\$:	
	EE		15	A0	E9	BLBC	21(RSTPTR), 1\$:	2889
					04	RET		:	
			50	D4	00026	CLRL	R0	:	2897
			04		00028	RET		:	

; Routine Size: 41 bytes, Routine Base: DBG\$CODE + 1103

```
2781 1 GLOBAL ROUTINE DBG$STA_NOEVALBIT(SYMID) =
2782 1
2783 1 FUNCTION
2784 1 This routine determines whether the DST$V_MS_NOEVAL bit is set in the
2785 1 Value Spec for a specified symbol. This bit is used by PL/I to suppress
2786 1 re-evaluation of Value Specs when such Value Specs can have side effects
2787 1 (as is the case for certain kinds of BASED variables). The side effects
2788 1 are acceptable when such a symbol is initially examined, but not when
2789 1 the symbol is reexamined via the dot pseudosymbol. Thus, when dot is
2790 1 bound to a symbol with the DST$V_MS_NOEVAL bit set in its Value Spec,
2791 1 that Value Spec is not reevaluated. The PL/I-specific code makes this
2792 1 check, but this routine returns the value of the bit.
2793 1
2794 1 The DST$V_MS_NOEVAL bit can only occur in a Value Spec containing a
2795 1 Materialization Spec. If the Value Spec does not have that form, this
2796 1 routine always returns FALSE--the bit is treated as not set.
2797 1
2798 1 INPUTS
2799 1 SYMID - The SYMID of the symbol whose DST$V_MS_NOEVAL bit is to be
2800 1 interrogated.
2801 1
2802 1 OUTPUTS
2803 1 The routine returns TRUE if the DST$V_MS_NOEVAL bit is set in the
2804 1 symbol's value spec. If the bit is not set or if the bit
2805 1 is not present at all in the symbol's value spec, FALSE
2806 1 is returned.
2807 1
2808 1
2809 2 BEGIN
2810 2
2811 2 MAP
2812 2 SYMID: REF RST$ENTRY; ! Pointer to input symbol's RST entry
2813 2
2814 2 LOCAL
2815 2 DSTPTR: REF DST$RECORD; ! Pointer to symbol's DST record.
2816 2 MSPTR: REF DST$MATER_SPEC; ! Pointer to DST Materialization Spec
2817 2 VSPTR: REF DST$VAL_SPEC; ! Pointer to DST Value Spec
2818 2
2819 2
2820 2 ! Determine what kind of RST entry SYMID identifies and act accordingly.
2821 2
2822 2 (CASE SYMID[RST$B_KIND] FROM RST$K_KIND_MINIMUM TO RST$K_KIND_MAXIMUM OF
2823 2 SET
2824 2
2825 2
2826 2
2827 2 ! For anything but Data and Type Component symbols, return FALSE. These
2828 2 symbols do not have value specs containing a DST$V_MS_NOEVAL bit.
2829 2
2830 2 [RST$K_ROUTINE, RST$K_BLOCK,
2831 2 RST$K_ENTRY, RST$K_LABEL,
2832 2 RST$K_LINE, RST$K_TYPE]:
2833 2 RETURN FALSE;
2834 2
2835 2
2836 2 ! For Data and Type Components, do nothing here--we handle them below.
2837 2
```



```
2838 2955 [RST$K_DATA, RST$K_TYPCOMP]:
2839 2956 0;
2840 2957
2841 2958
2842 2959 ! For everything else (including Module), signal an internal error.
2843 2960
2844 2961 [INRANGE, OVRANGE]:
2845 2962 $DBG_ERROR('RSTACCESS\NOEVALBIT');
2846 2963
2847 2964 TES;
2848 2965
2849 2966
2850 2967 ! For the items not yet handled (i.e., for Data), we determine the type of
2851 2968 ! DST record which holds the value specification and act accordingly.
2852 2969
2853 2970 DSTPTR = .SYMID[RST$L_DSTPTR];
2854 2971 CASE .DSTPTR[DST$B_TYPE] FROM 0 TO 255 OF
2855 2972 SET
2856 2973
2857 2974
2858 2975 ! Handle the DST records which can conceivably have Materialization
2859 2976 ! Specs and thus the DST$V_MS_NOEVAL bit. If the bit exists, return
2860 2977 ! its value; otherwise return FALSE.
2861 2978
2862 2979 [DSC$K_DTYPE_LOWEST TO DSC$K_DTYPE_HIGHEST,
2863 2980 DST$K_BOOL, DST$K_SEPTYP, DST$K_LBLORLIT,
2864 2981 DST$K_RECBEG, DST$K_ENUMELT]:
2865 2982 BEGIN
2866 2983
2867 2984
2868 2985 ! Indirect through any Trailing Value Specs to get to the symbol's
2869 2986 ! Value Spec. If this Value Spec cannot have a Materialization
2870 2987 ! Spec, return FALSE right away.
2871 2988
2872 2989 VSPTR = DSTPTR[DST$B_VFLAGS];
2873 2990 WHILE .VSPTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_TVS DO
2874 2991 VSPTR = VSPTR[DST$A_VS_TVS_BASE] + .VSPTR[DST$L_VS_TVS_OFFSET];
2875 2992
2876 2993 IF .VSPTR[DST$B_VS_VFLAGS] NEQ DST$K_VS_FOLLOWS THEN RETURN FALSE;
2877 2994
2878 2995
2879 2996 ! If this is a Static or Dynamic DST$K_VS_FOLLOWS type Value Spec,
2880 2997 ! return the DST$V_MS_NOEVAL bit from the Materialization Spec.
2881 2998
2882 2999 IF (.VSPTR[DST$B_VS_ALLOC] EQL DST$K_VS_ALLOC_STAT) OR
2883 3000 (.VSPTR[DST$B_VS_ALLOC] EQL DST$K_VS_ALLOC_DYN)
2884 3001 THEN
2885 3002 BEGIN
2886 3003 MSPTR = VSPTR[DST$A_VS_MATSPEC];
2887 3004 RETURN .MSPTR[DST$V_MS_NOEVAL];
2888 3005 END;
2889 3006
2890 3007
2891 3008 ! Any other value in the DST$B_VS_ALLOC field is an error.
2892 3009
2893 3010 SIGNAL(DBG$_INV DSTREC);
2894 3011
```

```

: 2895      3012 2      END;
: 2896      3013 2
: 2897      3014 2
: 2898      3015 2      ! For all DST records which cannot have Materialization Specs in their
: 2899      3016 2      ! Value Specs, fall through to return FALSE at the end of the routine.
: 2900      3017 2
: 2901      3018 2      [INRANGE]:
: 2902      3019 2      0;
: 2903      3020 2
: 2904      3021 2      TES;
: 2905      3022 2
: 2906      3023 2
: 2907      3024 2      ! Return FALSE. If we got here, there is no Materialization Spec.
: 2908      3025 2
: 2909      3026 2      RETURN FALSE;
: 2910      3027 2
: 2911      3028 1      END;

```

```

56 45 4F 4E 5C 53 53 45 43 43 41 54 53 52 13 000AC P.AAQ: .PSECT DBG$PLIT,NOWRT, SHR, PIC,0
54 49 42 4C 41 000BB .ASCII <19>\RSTACCESS\<92>\NOEVALBIT\

```

```

                                000C 00000
                                53 00000000G 00 9E 00002
                                52 04 AC D0 00009
                                00 14 A2 8F 0000D
0270      0D      001C      001C      00012 1$: .ENTRY DBG$STA NOEVALBIT, Save R2,R3
0270      0270      0270      0001A .MOVAB LIB$SIGNAL, R3
001C      002D      001C      0270      00022 .MOVL SYMID, R2
                                001C      001C      0002A .CASEB 20(R2), #0, #13
                                .WORD 2$-1$, -
                                2$-1$, -
                                10$-1$, -
                                10$-1$, -
                                10$-1$, -
                                10$-1$, -
                                3$-1$, -
                                10$-1$, -
                                10$-1$, -
                                2$-1$, -
                                3$-1$, -
                                2$-1$, -
                                2$-1$, -
                                2$-1$, -
                                00000000' EF 9F 0002E 2$: .PUSHAB P.AAQ
                                01 DD 00034 .PUSHL #1
00028362 8F DD 00036 .PUSHL #164706
                                63 03 FB 0003C .CALLS #3, LIB$SIGNAL
                                50 0C A2 D0 0003F 3$: .MOVL 12(R2), DSTPTR
                                00 01 A0 8F 00043 .CASEB 1(DSTPTR), #0, #255
0200      FF 8F      0200      0239      00049 4$: .WORD 10$-4$, -
0200      0200      0200      0200      00051 5$-4$, -
0200      0200      0200      0200      00059 5$-4$, -
0200      0200      0200      0200      00061 5$-4$, -

```

2898

2940

2962

2970

2971

0200	0200	0200	0200	00069	58-48,-
0200	0200	0200	0200	00071	58-48,-
0200	0200	0200	0200	00079	58-48,-
0200	0200	0200	0200	00081	58-48,-
0200	0200	0200	0200	00089	58-48,-
0239	0239	0200	0200	00091	58-48,-
0239	0239	0239	0239	00099	58-48,-
0239	0239	0239	0239	000A1	58-48,-
0239	0239	0239	0239	000A9	58-48,-
0239	0239	0239	0239	000B1	58-48,-
0239	0239	0239	0239	000B9	58-48,-
0239	0239	0239	0239	000C1	58-48,-
0239	0239	0239	0239	000C9	58-48,-
0239	0239	0239	0239	000D1	58-48,-
0239	0239	0239	0239	000D9	58-48,-
0239	0239	0239	0239	000E1	58-48,-
0239	0239	0239	0239	000E9	58-48,-
0239	0239	0239	0239	000F1	58-48,-
0239	0239	0239	0239	000F9	58-48,-
0239	0239	0239	0239	00101	58-48,-
0239	0239	0239	0239	00109	58-48,-
0239	0239	0239	0239	00111	58-48,-
0239	0239	0239	0239	00119	58-48,-
0239	0239	0239	0239	00121	58-48,-
0239	0239	0239	0239	00129	58-48,-
0239	0239	0239	0239	00131	58-48,-
0239	0239	0239	0239	00139	58-48,-
0239	0239	0239	0239	00141	58-48,-
0239	0239	0239	0239	00149	58-48,-
0239	0239	0239	0239	00151	58-48,-
0239	0239	0239	0239	00159	58-48,-
0239	0239	0239	0239	00161	58-48,-
0239	0239	0239	0239	00169	58-48,-
0239	0239	0239	0239	00171	58-48,-
0239	0239	0239	0239	00179	108-48,-
0239	0200	0239	0239	00181	108-48,-
0200	0239	0239	0239	00189	108-48,-
0239	0239	0239	0200	00191	108-48,-
0200	0239	0239	0239	00199	108-48,-
0239	0239	0239	0239	001A1	108-48,-
0239	0239	0239	0239	001A9	108-48,-
0239	0239	0239	0239	001B1	108-48,-
0239	0200	0239	0239	001B9	108-48,-
0239	0239	0239	0239	001C1	108-48,-
0239	0239	0239	0239	001C9	108-48,-
0239	0239	0239	0239	001D1	108-48,-
0239	0239	0239	0239	001D9	108-48,-
0239	0239	0239	0239	001E1	108-48,-
0239	0239	0239	0239	001E9	108-48,-
0239	0239	0239	0239	001F1	108-48,-
0239	0239	0239	0239	001F9	108-48,-
0239	0239	0239	0239	00201	108-48,-
0239	0239	0239	0239	00209	108-48,-
0239	0239	0239	0239	00211	108-48,-
0239	0239	0239	0239	00219	108-48,-
0239	0239	0239	0239	00221	108-48,-
0239	0239	0239	0239	00229	108-48,-

.....

RV

RSTACCESS
V04-000

```

E 8
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

```

Page 87
(17)

RV

[illegible]

.....

RSTACCESS
V04-000

```
F 8
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1
```

Page 88
(17)

[illegible]

```
G 8
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACKACCESS.B32:1
```

Page 89
(17)

PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

```
; Routine Size: 645 bytes,   Routine Base: DBG$CODE + 112C
```

```

: 2913 3029 1 GLOBAL ROUTINE DBG$STA_NUMBERED_SCOPE(SCOPE_NUMBER, MODRSTPTR, SCOPE,
: 2914 3030 1                                     INVOCNUM): NOVALUE =
: 2915 3031 1
: 2916 3032 1 FUNCTION
: 2917 3033 1     This routine determines what scope corresponds to a given 'numbered'
: 2918 3034 1     scope at this point in the user program's execution. This scope is de-
: 2919 3035 1     termined by looking SCOPE_NUMBER levels down in the VAX CALL-stack and
: 2920 3036 1     picking up the PC value in that call frame. The Program Static Address
: 2921 3037 1     Table (SAT) is searched for this PC value to find the containing module,
: 2922 3038 1     and after that the module's SAT is searched if the module is marked as
: 2923 3039 1     SET. The module's RST is built if not already present. The search is
: 2924 3040 1     successful if a Routine RST Entry or a Lexical Block RST Entry is found
: 2925 3041 1     whose address range contains the PC value.
: 2926 3042 1
: 2927 3043 1 INPUTS
: 2928 3044 1     SCOPE_NUMBER - The number of the 'numbered scope' to be located. This
: 2929 3045 1                     number is zero for the current scope, i.e. the scope where
: 2930 3046 1                     the PC is located at present, and it is N for the scope which
: 2931 3047 1                     contains the PC N levels down in the VAX CALL-stack.
: 2932 3048 1
: 2933 3049 1     MODRSTPTR - The address of a longword location to receive a pointer to
: 2934 3050 1                     the Module RST Entry for the numbered scope.
: 2935 3051 1
: 2936 3052 1     SCOPE - The address of a longword location to receive a pointer to the
: 2937 3053 1                     Routine or Lexical Block RST Entry which defines the numbered
: 2938 3054 1                     scope.
: 2939 3055 1
: 2940 3056 1     INVOCNUM - The address of a longword location to receive the correspond-
: 2941 3057 1                     ing invocation number.
: 2942 3058 1
: 2943 3059 1 OUTPUTS
: 2944 3060 1     MODRSTPTR - A pointer to the numbered scope's Module RST Entry is
: 2945 3061 1                     returned to MODRSTPTR. If the scope cannot be found, a
: 2946 3062 1                     zero is returned to MODRSTPTR.
: 2947 3063 1
: 2948 3064 1     SCOPE - A pointer to the RST entry of the routine or lexical block
: 2949 3065 1                     which constitutes the numbered scope is returned to SCOPE.
: 2950 3066 1                     If the scope cannot be found, a zero is returned to SCOPE.
: 2951 3067 1
: 2952 3068 1     INVOCNUM - The invocation number of the scope is returned to INVOCNUM.
: 2953 3069 1
: 2954 3070 1     No value is returned.
: 2955 3071 1
: 2956 3072 1
: 2957 3073 2 BEGIN
: 2958 3074 2
: 2959 3075 2 MAP
: 2960 3076 2     MODRSTPTR: REF VECTOR[1],           ! Pointer to longword to receive the
: 2961 3077 2                                     ! Module RST Entry pointer
: 2962 3078 2     SCOPE: REF VECTOR[1],           ! Pointer to longword to receive the
: 2963 3079 2                                     ! numbered scope RST pointer
: 2964 3080 2     INVOCNUM: REF VECTOR[1];       ! Pointer to longword to receive the
: 2965 3081 2                                     ! scope's invocation number
: 2966 3082 2
: 2967 3083 2 LOCAL
: 2968 3084 2     FRAMEPTR: REF BLOCK[.BYTE],         ! Pointer to stack CALL frames
: 2969 3085 2     MODPTR: REF RST$ENTRY,             ! Pointer to scope's Module RST Entry
```



```

: 2970      3086 2      PCVAL
: 2971      3087 2      REGPTR: REF VECTOR[.LONG],
: 2972      3088 2      REGVEC: VECTOR[17,.LONG],
: 2973      3089 2
: 2974      3090 2      ROUTPTR: REF RST$ENTRY,
: 2975      3091 2
: 2976      3092 2      RPTR: REF RST$ENTRY,
: 2977      3093 2      RSTPTR: REF RST$ENTRY,
: 2978      3094 2      RUNFRAME_PTR,
: 2979      3095 2
: 2980      3096 2
: 2981      3097 2      SATPTR: REF SAT$ENTRY;
: 2982      3098 2
: 2983      3099 2
: 2984      3100 2
: 2985      3101 2
: 2986      3102 2      ! Return zeroes (no find) to MODRSTPTR and SCOPE initially.
: 2987      3103 2
: 2988      3104 2      MODRSTPTR[0] = 0;
: 2989      3105 2      SCOPE[0] = 0;
: 2990      3106 2
: 2991      3107 2
: 2992      3108 2      ! Pick up the current Program Counter value from the user's run frame.
: 2993      3109 2      ! Then search through the CALL frames on the stack until the desired run
: 2994      3110 2      ! frame (and thus PC value) is reached. If the CALL stack ends before
: 2995      3111 2      ! then, return with MODRSTPTR and SCOPE containing zeroes.
: 2996      3112 2
: 2997      3113 2      PCVAL = .DBG$RUNFRAME[DBG$USER_PC];
: 2998      3114 2      IF .PCVAL EQL 0 THEN RETURN;
: 2999      3115 2      FRAMEPTR = .DBG$RUNFRAME[DBG$USER_FP];
: 3000      3116 2      RUNFRAME_PTR = .DBG$RUNFRAME[DBG$NEXT_LINK];
: 3001      3117 2      INCR I FROM 1 TO .SCOPE_NUMBER DO
: 3002      3118 3      BEGIN
: 3003      3119 4      IF (.FRAMEPTR[SFS$HANDLER] EQL DBG$FINAL_HANDL) OR (.PCVAL EQL 0)
: 3004      3120 3      THEN
: 3005      3121 3      RETURN;
: 3006      3122 3
: 3007      3123 3      GET REGISTER VALUES(.FRAMEPTR, RUNFRAME_PTR, REGVEC);
: 3008      3124 3      REGPTR = .REGVEC[15];
: 3009      3125 3      PCVAL = .REGPTR[0];
: 3010      3126 3      REGPTR = .REGVEC[13];
: 3011      3127 3      FRAMEPTR = .REGPTR[0];
: 3012      3128 2      END;
: 3013      3129 2
: 3014      3130 2
: 3015      3131 2      ! Search the Program Static Address Table (SAT) for the module which con-
: 3016      3132 2      ! tains the PC value we found. If we don't find such a module, return
: 3017      3133 2      ! with MODRSTPTR and SCOPE containing zeroes.
: 3018      3134 2
: 3019      3135 2      SATPTR = .SAT$START_ADDR;
: 3020      3136 2      WHILE TRUE DO
: 3021      3137 2      BEGIN
: 3022      3138 2      IF .SATPTR EQL 0 THEN RETURN;
: 3023      3139 2      IF .PCVAL GEQ .SATPTR[SAT$L_START] AND .PCVAL LEQ .SATPTR[SAT$L_END]
: 3024      3140 2      THEN
: 3025      3141 2      EXITLOOP;
: 3026      3142 2
```

```

: 3027      3143 3      SATPTR = .SATPTR[SAT$L_FLINK];
: 3028      3144 2      END;
: 3029      3145 2
: 3030      3146 2
: 3031      3147 2      ! We found the module. If the module is SET, search its SAT chain for the
: 3032      3148 2      ! inner-most lexical entity containing the PC value.
: 3033      3149 2
: 3034      3150 2      MODPTR = .SATPTR[SAT$L_RSTPTR];
: 3035      3151 2      IF NOT .MODPTR[RST$V_MODSET] THEN RETURN;
: 3036      3152 2      IF NOT .MODPTR[RST$V_MOD_IN_RST]
: 3037      3153 2      THEN
: 3038      3154 2          DBG$RST_BUILD(.MODPTR, FALSE);
: 3039      3155 2      RSTPTR = 0;
: 3040      3156 2      SATPTR = .MODPTR[RST$L_SAT_PTR];
: 3041      3157 2      WHILE TRUE DO
: 3042      3158 3          BEGIN
: 3043      3159 3              IF .SATPTR EQL 0 THEN EXITLOOP;
: 3044      3160 3              IF .SATPTR[SAT$L_START] GTR .PCVAL THEN EXITLOOP;
: 3045      3161 3              IF .SATPTR[SAT$L_END] GEQ .PCVAL
: 3046      3162 3              THEN
: 3047      3163 4                  BEGIN
: 3048      3164 4                      RPTR = .SATPTR[SAT$L_RSTPTR];
: 3049      3165 4
: 3050      3166 4
: 3051      3167 4                      ! If this static item is not a routine or block, ignore it.
: 3052      3168 4                      !
: 3053      3169 4                      IF (.RPTR[RST$B_KIND] NEQ RST$K_ROUTINE) AND
: 3054      3170 5                          (.RPTR[RST$B_KIND] NEQ RST$K_BLOCK)
: 3055      3171 4                      THEN
: 3056      3172 4                          0
: 3057      3173 4
: 3058      3174 4
: 3059      3175 4                      ! It is a lexical entity. If it is the first one we have found,
: 3060      3176 4                      ! save its RST pointer in RSTPTR.
: 3061      3177 4                      !
: 3062      3178 4                      ELSE IF .RSTPTR EQL 0
: 3063      3179 4                      THEN
: 3064      3180 4                          RSTPTR = .SATPTR[SAT$L_RSTPTR]
: 3065      3181 4
: 3066      3182 4
: 3067      3183 4                      ! Otherwise, make sure it is the inner-most lexical entity so far.
: 3068      3184 4                      ! If not, ignore it.
: 3069      3185 4                      !
: 3070      3186 4                      ELSE
: 3071      3187 5                          BEGIN
: 3072      3188 5                              WHILE .RPTR[RST$B_KIND] NEQ RST$K_MODULE DO
: 3073      3189 6                                  BEGIN
: 3074      3190 6                                      IF .RPTR EQL .RSTPTR
: 3075      3191 6                                      THEN
: 3076      3192 7                                          BEGIN
: 3077      3193 7                                              RSTPTR = .SATPTR[SAT$L_RSTPTR];
: 3078      3194 7                                              EXITLOOP;
: 3079      3195 6                                          END;
: 3080      3196 6
: 3081      3197 6                      RPTR = .RPTR[RST$L_UPSCOPEPTR];
: 3082      3198 5                      END;
: 3083      3199 5
```

```

: 3084      3200      4      END;
: 3085      3201      4
: 3086      3202      3      END;
: 3087      3203      3
: 3088      3204      3      SATPTR = .SATPTR[SAT$$_LINK];
: 3089      3205      2      END;
: 3090      3206      2
: 3091      3207      2
: 3092      3208      2      ! If we did not find the containing lexical entity, return with MODRSTPTR
: 3093      3209      2      ! and SCOPE containing zeroes.
: 3094      3210      2
: 3095      3211      2      IF .RSTPTR EQL 0 THEN RETURN;
: 3096      3212      2
: 3097      3213      2
: 3098      3214      2      ! We found the scope successfully. Return the proper RST pointers to
: 3099      3215      2      ! MODRSTPTR and SCOPE.
: 3100      3216      2
: 3101      3217      2      MODRSTPTR[0] = .MODPTR;
: 3102      3218      2      SCOPE[0] = .RSTPTR;
: 3103      3219      2
: 3104      3220      2
: 3105      3221      2      ! Now search the CALL stack again to determine what the invocation number is
: 3106      3222      2      ! for the routine which constitutes or immediately contains the scope.
: 3107      3223      2
: 3108      3224      2      INVOCNUM[0] = 0;
: 3109      3225      2      ROUTPTR = .RSTPTR;
: 3110      3226      2      WHILE .ROUTPTR[RST$$_KIND] NEQ RST$$_ROUTINE DO
: 3111      3227      3      BEGIN
: 3112      3228      3      IF .ROUTPTR[RST$$_KIND] EQL RST$$_MODULE THEN RETURN;
: 3113      3229      3      ROUTPTR = .ROUTPTR[RST$$_UPSCOPEPTR];
: 3114      3230      2      END;
: 3115      3231      2
: 3116      3232      2      PCVAL = .DBG$$_RUNFRAME[DBG$$_USER_PC];
: 3117      3233      2      FRAMEPTR = .DBG$$_RUNFRAME[DBG$$_USER_FP];
: 3118      3234      2      RUNFRAME_PTR = .DBG$$_RUNFRAME[DBG$$_NEXT_LINK];
: 3119      3235      2      INCR I FROM 1 TO .SCOPE_NUMBER DO
: 3120      3236      3      BEGIN
: 3121      3237      3      IF (.PCVAL GEQ .ROUTPTR[RST$$_STARTADDR]) AND
: 3122      3238      4      (.PCVAL LEQ .ROUTPTR[RST$$_ENDADDR])
: 3123      3239      3      THEN
: 3124      3240      3      INVOCNUM[0] = .INVOCNUM[0] + 1;
: 3125      3241      3
: 3126      3242      3      GET REGISTER VALUES(.FRAMEPTR, RUNFRAME_PTR, REGVEC);
: 3127      3243      3      REGPTR = .REGVEC[15];
: 3128      3244      3      PCVAL = .REGPTR[0];
: 3129      3245      3      REGPTR = .REGVEC[13];
: 3130      3246      3      FRAMEPTR = .REGPTR[0];
: 3131      3247      2      END;
: 3132      3248      2
: 3133      3249      2
: 3134      3250      2      ! We are all done. Now return.
: 3135      3251      2
: 3136      3252      2      RETURN;
: 3137      3253      2
: 3138      3254      1      END;
```

			00FC 00000	.ENTRY	DBG\$STA_NUMBERED_SCOPE, Save R2,R3,R4,R5,-	
					R6,R7	3029
57	00000000G	00	9E 00002	MOVAB	DBG\$RUNFRAME+64, R7	
5E	B8	AE	9E 00009	MOVAB	-72(SP), SP	
	08	BC	D4 0000D	CLRL	@MODRSTPTR	3104
	0C	BC	D4 00010	CLRL	@SCOPE	3105
54		67	D0 00013	MOVL	DBG\$RUNFRAME+64, PCVAL	3113
		43	13 00016	BEQL	3\$	3114
56	F8	A7	D0 00018	MOVL	DBG\$RUNFRAME+56, FRAMEPTR	3115
6E	C0	A7	D0 0001C	MOVL	DBG\$RUNFRAME, RUNFRAME_PTR	3116
		52	D4 00020	CLRL	1	3117
		2B	11 00022	BRB	2\$	
50	00000000G	00	9E 00024 1\$:	MOVAB	DBG\$FINAL_HANDL, R0	3119
50		66	D1 0002B	CMPL	(FRAMEPTR), R0	
		2B	13 0002E	BEQL	3\$	
		54	D5 00030	TSTL	PCVAL	
		27	13 00032	BEQL	3\$	
	04	AE	9F 00034	PUSHAB	REGVEC	3123
	04	AE	9F 00037	PUSHAB	RUNFRAME_PTR	
		56	DD 0003A	PUSHL	FRAMEPTR	
0000V	CF	03	FB 0003C	CALLS	#3, GET_REGISTER_VALUES	
	55	40	AE D0 00041	MOVL	REGVEC+80, REGPTR	3124
	54		65 D0 00045	MOVL	(REGPTR), PCVAL	3125
	55	38	AE D0 00048	MOVL	REGVEC+52, REGPTR	3126
	56		65 D0 0004C	MOVL	(REGPTR), FRAMEPTR	3127
D0		52	04 AC F3 0004F 2\$:	AOBLEQ	SCOPE_NUMBER, 1, 1\$	3117
		52	00000000G	MOVL	SAT\$START_ADDR, SATPTR	3135
		70	13 0005B 3\$:	BEQL	15\$	3138
04	A2	54	D1 0005D	CMPL	PCVAL, 4(SATPTR)	3139
		06	19 00061	BLSS	4\$	
08	A2	54	D1 00063	CMPL	PCVAL, 8(SATPTR)	
		05	15 00067	BLEQ	5\$	
	52	62	D0 00069 4\$:	MOVL	(SATPTR), SATPTR	3143
		ED	11 0006C	BRB	3\$	3136
	53	0C	A2 D0 0006E 5\$:	MOVL	12(SATPTR), MODPTR	3150
	01	28	A3 E8 00072	BLBS	40(MODPTR), 6\$	3151
			04 00076	RET		
08	28	A3	01 E0 00077 6\$:	BBS	#1, 40(MODPTR), 7\$	3152
		7E	D4 0007C	CLRL	-(SP)	3154
		53	DD 0007E	PUSHL	MODPTR	
00000000G	00	02	FB 00080	CALLS	#2, DBG\$RST_BUILD	
		51	D4 00087 7\$:	CLRL	RSTPTR	3155
	52	18	A3 D0 00089	MOVL	24(MODPTR), SATPTR	3156
		3C	13 0008D 8\$:	BEQL	14\$	3159
	54	04	A2 D1 0008F	CMPL	4(SATPTR), PCVAL	3160
		36	14 00093	BGTR	14\$	
	54	08	A2 D1 00095	CMPL	8(SATPTR), PCVAL	3161
		2B	19 00099	BLSS	13\$	
	50	0C	A2 D0 0009B	MOVL	12(SATPTR), RPTR	3164
	02	14	A0 91 0009F	CMPL	20(RPTR), #2	3169
		06	13 000A3	BEQL	9\$	
	03	14	A0 91 000A5	CMPL	20(RPTR), #3	3170
		1B	12 000A9	BNEQ	13\$	
		51	D5 000AB 9\$:	TSTL	RSTPTR	3178

		0B	13	000AD		BEQL	11\$		
01		14	A0	91 000AF	10\$:	CMPB	20(RPTR), #1		3188
			11	13 000B3		BEQL	13\$		
51			50	D1 000B5		CMPL	RPTR, RSTPTR		3190
			06	12 000B8		BNEQ	12\$		
51		0C	A2	D0 000BA	11\$:	MOVL	12(SATPTR), RSTPTR		3193
			06	11 000BE		BRB	13\$		3192
50		10	A0	D0 000C0	12\$:	MOVL	16(RPTR), RPTR		3197
			E9	11 000C4		BRB	10\$		3188
52			62	D0 000C6	13\$:	MOVL	(SATPTR), SATPTR		3204
			C2	11 000C9		BRB	8\$		3157
			51	D5 000CB	14\$:	TSTL	RSTPTR		3211
			5E	13 000CD	15\$:	BEQL	21\$		
08	BC		53	D0 000CF		MOVL	MODPTR, @MODRSTPTR		3217
0C	BC		51	D0 000D3		MOVL	RSTPTR, @SCOPE		3218
		10	BC	D4 000D7		CLRL	@INVOCNUM		3224
			51	D0 000DA		MOVL	RSTPTR, ROUTPTR		3225
		14	A2	91 000DD	16\$:	CMPB	20(ROUTPTR), #2		3226
			0C	13 000E1		BEQL	17\$		
		14	A2	91 000E3		CMPB	20(ROUTPTR), #1		3228
			44	13 000E7		BEQL	21\$		
		10	A2	D0 000E9		MOVL	16(ROUTPTR), ROUTPTR		3229
			EE	11 000ED		BRB	16\$		3226
			67	D0 000EF	17\$:	MOVL	DBG\$RUNFRAME+64, PCVAL		3232
		F8	A7	D0 000F2		MOVL	DBG\$RUNFRAME+56, FRAMEPTR		3233
		C0	A7	D0 000F6		MOVL	DBG\$RUNFRAME, RUNFRAME_PTR		3234
			53	D4 000FA		CLRL	1		3237
			2A	11 000FC		BRB	20\$		
18	A2		54	D1 000FE	18\$:	CMPL	PCVAL, 24(ROUTPTR)		
			09	19 00102		BLSS	19\$		
1C	A2		54	D1 00104		CMPL	PCVAL, 28(ROUTPTR)		3238
			03	14 00108		BGTR	19\$		
		10	BC	D6 0010A		INCL	@INVOCNUM		3240
		04	AE	9F 0010D	19\$:	PUSHAB	REGVEC		3242
		04	AE	9F 00110		PUSHAB	RUNFRAME_PTR		
			56	DD 00113		PUSHL	FRAMEPTR		
0000V	CF		03	FB 00115		CALLS	#3, GET REGISTER VALUES		
	55	40	AE	D0 0011A		MOVL	REGVEC+80, REGPTR		3243
	54		65	D0 0011E		MOVL	(REGPTR), PCVAL		3244
	55	38	AE	D0 00121		MOVL	REGVEC+52, REGPTR		3245
	56		65	D0 00125		MOVL	(REGPTR), FRAMEPTR		3246
D1	53	04	AC	F3 00128	20\$:	A0BLEQ	SCOPE_NUMBER, 1, 18\$		3235
			04	0012D	21\$:	RET			3254

; Routine Size: 302 bytes, Routine Base: DBG\$CODE + 13B1

```
3140 3255 1 GLOBAL ROUTINE DBG$STA_RECORD_COMPONENT(RECSYMID, INDEX) =
3141 3256 1
3142 3257 1 FUNCTION
3143 3258 1 This routine returns the SYMID of the N-th record component of a record
3144 3259 1 data object. It accepts as input a pointer to a Data RST Entry of a
3145 3260 1 record ("structure") data object and an index ('N') into the list of
3146 3261 1 record components for the record. This routine is used mainly to find
3147 3262 1 the logical successor or predecessor of a record component of a known
3148 3263 1 index into the record component list. In other words, if the current
3149 3264 1 location is the N-th component of a record, its predecessor is the N-1
3150 3265 1 and its successor the N+1 component of the record. This routine returns
3151 3266 1 a pointer to the Data RST Entry for such a component.
3152 3267 1
3153 3268 1 To accomplish this, the INDEX-th record component is looked up in the
3154 3269 1 component list in the record's Data Type RST Entry. This gives a
3155 3270 1 pointer to the component's Type Component RST Entry. A new Data Item
3156 3271 1 RST Entry is then build from the information in the Type Component
3157 3272 1 RST Entry. This new entry is put on the Temporary RST Entry List and
3158 3273 1 its address is returned as the component SYMID.
3159 3274 1
3160 3275 1 INPUTS
3161 3276 1 RECSYMID - The SYMID of the Record data object whose INDEX-th component
3162 3277 1 is to be returned.
3163 3278 1
3164 3279 1 INDEX - The index of the desired component into the record component
3165 3280 1 list for RECSYMID. The first component of a record has the
3166 3281 1 INDEX value of 1.
3167 3282 1
3168 3283 1 OUTPUTS
3169 3284 1 The SYMID of the INDEX-th component of RECSYMID is returned as the
3170 3285 1 routine value. If INDEX is out of range (no such component
3171 3286 1 number), this routine returns zero.
3172 3287 1
3173 3288 1
3174 3289 2 BEGIN
3175 3290 2
3176 3291 2 MAP
3177 3292 2 RECSYMID: REF RST$ENTRY; ! SYMID of record data object
3178 3293 2
3179 3294 2 LOCAL
3180 3295 2 FCODE,
3181 3296 2 NEWRSIPTR: REF RST$ENTRY, ! Pointer to new Data Item RST Entry
3182 3297 2 RSTPTR: REF RST$ENTRY, ! Pointer to Type Component RST Entry
3183 3298 2 TYP COMPLST: REF VECTOR[.LONG], ! Pointer to type component list
3184 3299 2 TYPEPTR: REF RST$ENTRY; ! Pointer to record's Type RST Entry
3185 3300 2
3186 3301 2
3187 3302 2
3188 3303 2 ! Check that RECSYMID points to the Data Item RST Entry for a Record object.
3189 3304 2 ! If not, signal an internal DEBUG error.
3190 3305 2
3191 3306 2 IF .RECSYMID[RST$B_KIND] NEQ RST$K_DATA
3192 3307 2 THEN
3193 3308 2 $DBG_ERROR('RSTACCESS\RECORD_COMPONENT 10');
3194 3309 2
3195 3310 2 DBG$STA_SYMTYPE(.RECSYMID,FCODE,TYPEPTR);
3196 3311 2 IF .FCODE EQL RST$K_TYPE_ARRAY
```

```

3197 3312 2 THEN
3198 3313 2 BEGIN
3199 3314 2 LOCAL DSCADDR,NDIMS,DIMVECPTR,BITSIZE;
3200 3315 2 DBG$STA_TYP_ARRAY(.TYPEPTR,DSCADDR,TYPEPTR,NDIMS,DIMVECPTR,BITSIZE);
3201 3316 2 END;
3202 3317 2
3203 3318 2 IF .TYPEPTR[RST$B_FCODE] NEQ RST$K_TYPE_RECORD
3204 3319 2 THEN
3205 3320 2 $DBG_ERROR('RSTACCESS\RECORD_COMPONENT 20');
3206 3321 2
3207 3322 2
3208 3323 2 ! If the INDEX value is out of range, return zero to the caller. Otherwise,
3209 3324 2 ! pick up a pointer to the INDEX-th type Component RST Entry.
3210 3325 2
3211 3326 2 IF (.INDEX LEQ 0) OR (.INDEX GTR .TYPEPTR[RST$L_TYPCOMPNT]) THEN RETURN 0;
3212 3327 2 TYPCOMPLST = TYPEPTR[RST$A_TYPCOMPLST];
3213 3328 2 RSTPTR = .TYPCOMPLST[.INDEX - 1];
3214 3329 2 IF .RSTPTR[RST$B_KIND] NEQ RST$K_TYPCOMP
3215 3330 2 THEN
3216 3331 2 $DBG_ERROR('RSTACCESS\RECORD_COMPONENT 30');
3217 3332 2
3218 3333 2
3219 3334 2 ! Now construct a Data Item RST Entry from the Type Component RST Entry,
3220 3335 2 ! make its up-scope pointer point to the RECSYDID Data Item RST Entry,
3221 3336 2 ! and return the address (i.e., SYDID) of the new RST entry to the caller.
3222 3337 2 ! Note that the new RST entry is put on the Temporary RST Entry List.
3223 3338 2
3224 3339 2 NEWRSTPTR = DBG$GET MEMORY(RST$K_DATENTSIZ);
3225 3340 2 NEWRSTPTR[RST$L_HASH_FLINK] = .RST$TEMP_LIST;
3226 3341 2 RST$TEMP_LIST = .NEWRSTPTR;
3227 3342 2 NEWRSTPTR[RST$L_DSTPTR] = .RSTPTR[RST$L_DSTPTR];
3228 3343 2 NEWRSTPTR[RST$L_UPSCOPEPTR] = .RECSYDID;
3229 3344 2 NEWRSTPTR[RST$B_KIND] = RST$K_DATA;
3230 3345 2 NEWRSTPTR[RST$L_TYPEPTR] = .RSTPTR[RST$L_TYPEPTR];
3231 3346 2 RETURN .NEWRSTPTR;
3232 3347 2
3233 3348 1 END;
```

```

                                .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0
4F  43  45  52  5C  53  53  45  43  43  41  54  53  52  1D  000C0 P.AAR:  .ASCII  <29>\RSTACCESS\<92>\RECORD_COMPONENT 1\
    31  20  54  4E  45  4E  4F  50  4D  4F  43  5F  44  52  000CF
                                30  000DD
4F  43  45  52  5C  53  53  45  43  43  41  54  53  52  1D  000DE P.AAS:  .ASCII  <29>\RSTACCESS\<92>\RECORD_COMPONENT 2\
    32  20  54  4E  45  4E  4F  50  4D  4F  43  5F  44  52  000ED
                                30  000FB
4F  43  45  52  5C  53  53  45  43  43  41  54  53  52  1D  000FC P.AAT:  .ASCII  <29>\RSTACCESS\<92>\RECORD_COMPONENT 3\
    33  20  54  4E  45  4E  4F  50  4D  4F  43  5F  44  52  0010B
                                30  00119
                                .ASCII  \0\
```

```

                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0
                                007C 00000
                                .ENTRY  DBG$STA_RECORD_COMPONENT, Save R2,R3,R4,R5,-; 3255
```

56	00000000G	00	9E	00002	MOVAB	R6		
55	00000000'	EF	9E	00009	MOVAB	RST\$TEMP_LIST, R6		
54	00000000G	00	9E	00010	MOVAB	P.AAR, R5		
5E		18	C2	00017	MOVAB	LIB\$SIGNAL, R4		
53	04	AC	D0	0001A	SUBL2	#24, SP		
06	14	A3	91	0001E	MOVL	REC\$YMD, R3	3306	
		0D	13	00022	CMPB	20(R3), #6		
		55	DD	00024	BEQL	1\$		
		01	DD	00026	PUSHL	R5	3308	
	00028362	8F	DD	00028	PUSHL	#1		
64		03	FB	0002E	PUSHL	#164706		
	10	AE	9F	00031	CALLS	#3, LIB\$SIGNAL		
	04	AE	9F	00034	PUSHAB	TYPEPTR	3310	
		53	DD	00037	PUSHAB	FCODE		
00000000G	00	03	FB	00039	PUSHL	R3		
01		6E	D1	00040	CALLS	#3, DBG\$STA_SYMTYPE	3311	
		19	12	00043	CMPL	FCODE, #1		
	04	AE	9F	00045	BNEQ	2\$		
	0C	AE	9F	00048	PUSHAB	BITSIZE	3315	
	14	AE	9F	0004B	PUSHAB	DIMVECPTR		
	1C	AE	9F	0004E	PUSHAB	NDIMS		
	24	AE	9F	00051	PUSHAB	TYPEPTR		
	24	AE	DD	00054	PUSHAB	DSCADDR		
00000000G	00	06	FB	00057	PUSHL	TYPEPTR		
52	10	AE	D0	0005E	CALLS	#6, DBG\$STA_TYP_ARRAY		
07	18	A2	91	00062	MOVL	TYPEPTR, R2	3318	
		0E	13	00066	CMPB	24(R2), #7		
	1E	A5	9F	00068	BEQL	3\$		
		01	DD	0006B	PUSHAB	P.AAS	3320	
	00028762	8F	DD	0006D	PUSHL	#1		
64		03	FB	00073	PUSHL	#164706		
51	08	AC	D0	00076	CALLS	#3, LIB\$SIGNAL		
		45	15	0007A	MOVL	INDEX, R1	3326	
28	A2	51	D1	0007C	BLEQ	5\$		
		3F	14	00080	CMPL	R1, 40(R2)		
50	2C	A2	9E	00082	BGTR	5\$		
52	FC	A041	D0	00086	MOVAB	44(R2), TYP\$COMPLST	3327	
0A	14	A2	91	0008B	MOVL	-4(TYP\$COMPLST)[R1], RSTPTR	3328	
		0E	13	0008F	CMPB	20(RSTPTR), #10	3329	
	3C	A5	9F	00091	BEQL	4\$		
		01	DD	00094	PUSHAB	P.AAT	3331	
	00028362	8F	DD	00096	PUSHL	#1		
64		03	FB	0009C	PUSHL	#164706		
		07	DD	0009F	CALLS	#3, LIB\$SIGNAL		
00000000G	00	01	FB	000A1	PUSHL	#7	3339	
60		66	D0	000AB	CALLS	#1, DBG\$GET_MEMORY		
66		50	D0	000AB	MOVL	RST\$TEMP_LIST, (NEWSTPTR)	3340	
0C	A0	50	D0	000AB	MOVL	NEWSTPTR, RST\$TEMP_LIST	3341	
10	A0	53	D0	000B3	MOVL	12(RSTPTR), 12(NEWSTPTR)	3342	
14	A0	06	90	000B7	MOVL	R3, 16(NEWSTPTR)	3343	
18	A0	18	A2	000BB	MOVB	#6, 20(NEWSTPTR)	3344	
			04	000C0	MOVL	24(RSTPTR), 24(NEWSTPTR)	3345	
		50	D4	000C1	RET		3346	
		04	000C3	5\$:	CLRL	R0	3347	
					RET		3348	

; Routine Size: 196 bytes. Routine Base: DBG\$CODE + 14DF

RSTACCESS
V04-000

D 9
16-Sep-1984 02:48:17
14-Sep-1984 12:18:26

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]RSTACCESS.B32;1

Page 99
(19)

R
V

```
3235 3349 1 GLOBAL ROUTINE DBG$STA_RECORD_INDEX(RECSYMID, COMPSYMID) =
3236 3350 1
3237 3351 1 FUNCTION
3238 3352 1 This routine accepts the SYMID of a Record data object and the SYMID
3239 3353 1 of a component of that record and it returns the index of the component
3240 3354 1 in the record's component list. Both the Record and Component SYMIDs
3241 3355 1 should point to Data Item RST Entries (kind is RST$K_DATA). The re-
3242 3356 1 turned index starts at 1 so that the index of the first component of
3243 3357 1 the record is 1, the index of the second component is 2, and so forth.
3244 3358 1 This routine is used together with routine DBG$STA_RECORD_COMPONENT in
3245 3359 1 the processing of logical predecessors and successors.
3246 3360 1
3247 3361 1 If the COMPSYMID object is not a component of the RECSYMID object or if
3248 3362 1 the RECSYMID object is not of a Record type, an internal DEBUG error is
3249 3363 1 signalled.
3250 3364 1
3251 3365 1 The routine does its job by searching the Type Component List in the
3252 3366 1 Data Type RST Entry for the record type for a Type Component RST Entry
3253 3367 1 which has the same DST pointer as the COMPSYMID Data Item RST Entry.
3254 3368 1 When such an entry is found, its index in the list is returned.
3255 3369 1
3256 3370 1 INPUTS
3257 3371 1 RECSYMID - The SYMID of a Record ('structure') data object. Its kind
3258 3372 1 must be RST$K_DATA.
3259 3373 1
3260 3374 1 COMPSYMID - The SYMID of a component of the RECSYMID record. Its kind
3261 3375 1 must also be RST$K_DATA.
3262 3376 1
3263 3377 1 OUTPUTS
3264 3378 1 The index of the COMPSYMID data object in the record component list for
3265 3379 1 the RECSYMID data record. The first component has index 1.
3266 3380 1
3267 3381 1
3268 3382 2 BEGIN
3269 3383 2
3270 3384 2 MAP
3271 3385 2 RECSYMID: REF RST$ENTRY, : Pointer to Data RST Entry for record
3272 3386 2 COMPSYMID: REF RST$ENTRY; : Pointer to Data RST Entry for a com-
3273 3387 2 : ponent within the above record
3274 3388 2
3275 3389 2 LOCAL
3276 3390 2 FCODE,
3277 3391 2 RSTPTR: REF RST$ENTRY, : Pointer to Type Component RST Entry
3278 3392 2 TYPECOMPLST: REF VECTOR[.LONG], : Pointer to type component list in the
3279 3393 2 : Data Type RST Entry
3280 3394 2 TYPEPTR: REF RST$ENTRY; : Pointer to Type RST Entry for record
3281 3395 2
3282 3396 2
3283 3397 2
3284 3398 2 : Make sure RECSYMID points to a Data Item RST Entry for a record and that
3285 3399 2 : COMPSYMID points to a Data Item RST Entry. Get a pointer to the Type RST
3286 3400 2 : Entry for the record type.
3287 3401 2
3288 3402 2 IF (.RECSYMID[RST$B_KIND] NEQ RST$K_DATA) OR
3289 3403 2 (.COMPSYMID[RST$B_KIND] NEQ RST$K_DATA)
3290 3404 2 THEN
3291 3405 2 $DBG_ERROR('RSTACCESS\RECORD_INDEX 10');
```

```

: 3292      3406 2
: 3293      3407 2
: 3294      3408 3
: 3295      3409 3
: 3296      3410 3
: 3297      3411 3
: 3298      3412 3
: 3299      3413 2
: 3300      3414 2
: 3301      3415 2
: 3302      3416 2
: 3303      3417 3
: 3304      3418 3
: 3305      3419 2
: 3306      3420 2
: 3307      3421 2
: 3308      3422 2
: 3309      3423 2
: 3310      3424 2
: 3311      3425 2
: 3312      3426 2
: 3313      3427 3
: 3314      3428 3
: 3315      3429 3
: 3316      3430 2
: 3317      3431 2
: 3318      3432 3
: 3319      3433 2
: 3320      3434 2
: 3321      3435 2
: 3322      3436 2
: 3323      3437 2
: 3324      3438 1

DBG$STA_SYMTYPE(.RECSYMID,FCODE,TYPEPTR);
IF (.FCODE EQL RST$K_TYPE_ARRAY)
THEN
    BEGIN
        LOCAL DSCADDR,NDIMS,DIMVECPTR,BITSIZE;
        DBG$STA_TYP_ARRAY(.TYPEPTR,DSCADDR,TYPEPTR,NDIMS,DIMVECPTR,BITSIZE);
    END;

IF .TYPEPTR[RST$B_FCODE] NEQ RST$K_TYPE_RECORD
THEN
    $DBG_ERROR('RSTACCESS\RECORD_INDEX 20');

! Now loop through the record components for the RECSYMID Record Data Type.
! For each component, we get an index and a pointer to the corresponding
! Type Component RST Entry. If that Type Component RST Entry has the same
! DST pointer as COMPSYMID, we return that component's index.
TYPCOMPLST = TYPEPTR[RST$A_TYPCOMPLST];
INCR INDEX FROM 1 TO .TYPEPTR[RST$L_TYPCOMPNT] DO
    BEGIN
        RSTPTR = .TYPCOMPLST[.INDEX - 1];
        IF .RSTPTR[RST$L_DSTPTR] EQL .COMPSYMID[RST$L_DSTPTR] THEN RETURN .INDEX;
    END;

! We did not find COMPSYMID in the component list. Signal an error.
$DBG_ERROR('RSTACCESS\RECORD_INDEX 30');
RETURN 0;

END;
```

```

                                .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0
4F  43  45  52  5C  53  53  45  43  43  41  54  53  52  19  0011A P.AAU: .ASCII <25>\RSTACCESS\<92>\RECORD_INDEX 10\
: 30  31  20  58  45  44  4E  49  5F  44  52  00129
4F  43  45  52  5C  53  53  45  43  43  41  54  53  52  19  00134 P.AAV: .ASCII <25>\RSTACCESS\<92>\RECORD_INDEX 20\
: 30  32  20  58  45  44  4E  49  5F  44  52  00143
4F  43  45  52  5C  53  53  45  43  43  41  54  53  52  19  0014E P.AAW: .ASCII <25>\RSTACCESS\<92>\RECORD_INDEX 30\
: 30  33  20  58  45  44  4E  49  5F  44  52  0015D
```

```

                                007C 00000
56 00000000' EF 9E 00002
55 00000000G 00 9E 00009
5E          18 C2 00010
52          04 AC D0 00013
06          14 A2 91 00017
          0A 12 0001B
50          08 AC D0 0001D
```

```

                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0
                                .ENTRY  DBG$STA_RECORD_INDEX, Save R2,R3,R4,R5,R6 : 3349
MOVAB P.AAU, R6
MOVAB LIB$SIGNAL, R5
SUBL2 #24, SP
MOVL RECSYMID, R2 : 3402
CMPB 20(R2), #6
BNEQ 1$
MOVL COMPSYMID, R0 : 3403
```

06	14	A0	91	00021	CMPB	20(R0), #6	:	
		0D	13	00025	BEQL	2\$:	
		56	DD	00027	PUSHL	R6	:	3405
		01	DD	00029	PUSHL	#1	:	
	00028362	8F	DD	0002B	PUSHL	#164706	:	
65		03	FB	00031	CALLS	#3, LIB\$SIGNAL	:	
	10	AE	9F	00034	PUSHAB	TYPEPTR	:	3407
	04	AE	9F	00037	PUSHAB	FCODE	:	
		52	DD	0003A	PUSHL	R2	:	
00000000G	00	03	FB	0003C	CALLS	#3, DBG\$STA_SYMTYPE	:	
	01	6E	D1	00043	CMPL	FCODE, #1	:	3408
		19	12	00046	BNEQ	3\$:	
	04	AE	9F	00048	PUSHAB	BITSIZE	:	3412
	0C	AE	9F	0004B	PUSHAB	DIMVECPTR	:	
	14	AE	9F	0004E	PUSHAB	NDIMS	:	
	1C	AE	9F	00051	PUSHAB	TYPEPTR	:	
	24	AE	9F	00054	PUSHAB	DSCADDR	:	
	24	AE	DD	00057	PUSHL	TYPEPTR	:	
00000000G	00	06	FB	0005A	CALLS	#6, DBG\$STA_TYP_ARRAY	:	
	54	10	AE	DD	00061	MOVL	TYPEPTR, R4	3415
	07	18	A4	91	00065	CMPB	24(R4), #7	
		0E	13	00069	BEQL	4\$:	
	1A	A6	9F	0006B	PUSHAB	P.AAV	:	3417
		01	DD	0006E	PUSHL	#1	:	
	00028362	8F	DD	00070	PUSHL	#164706	:	
65		03	FB	00076	CALLS	#3, LIB\$SIGNAL	:	
	2C	A4	9E	00079	MOVAB	44(R4), TYPCOMPLST	:	3425
50		AC	D0	0007D	MOVL	COMPSYD, R2	:	3429
52	08	51	D4	00081	CLRL	INDEX	:	
		10	11	00083	BRB	6\$:	
	53	FC	A041	D0	00085	MOVL	-4(TYPCOMPLST)[INDEX], RSTPTR	3428
OC	A2	0C	A3	D1	0008A	CMPL	12(RSTPTR), 12(R2)	3429
		04	12	0008F	BNEQ	6\$:	
	50	51	D0	00091	MOVL	INDEX, R0	:	
		04	00094	RET			:	
EB	51	28	A4	F3	00095	AOBLEQ	40(R4), INDEX, 5\$	3426
		34	A6	9F	0009A	PUSHAB	P.AAW	3435
		01	DD	0009D	PUSHL	#1	:	
	00028362	8F	DD	0009F	PUSHL	#164706	:	
65		03	FB	000A5	CALLS	#3, LIB\$SIGNAL	:	
		50	D4	000AB	CLRL	R0	:	3436
		04	000AA	RET			:	3438

; Routine Size: 171 bytes, Routine Base: DBG\$CODE + 15A3

```
3326 3439 1 GLOBAL ROUTINE DBG$STA_REGISTER_NAME(REGDESCR) =
3327 3440 1
3328 3441 1 FUNCTION
3329 3442 1     This routine converts a Register Descriptor into a Counted ASCII name
3330 3443 1     for the corresponding register. A Register Descriptor is produced
3331 3444 1     from an absolute address by routine DBG$STA_ADDRESS_TO_REGDESCR if
3332 3445 1     the address points into the DBG$REG_VALUES register save area set up
3333 3446 1     by DBG$STA_SETCODETEXT. The Register Descriptor contains the register
3334 3447 1     number, a byte offset, and the scope number of the register described.
3335 3448 1     A register address is not a normal address, and should be printed as
3336 3449 1     a scope number or scope name followed by a register name. This routine
3337 3450 1     builds such a register name as a Counted ASCII string (for example
3338 3451 1     '2\XR5') and returns a pointer to that string.
3339 3452 1
3340 3453 1 INPUTS
3341 3454 1     REGDESCR - The Register Descriptor which describes the register name
3342 3455 1     to be generated.
3343 3456 1
3344 3457 1 OUTPUTS
3345 3458 1     A pointer to a Counted ASCII string containing the appropriate register
3346 3459 1     name is returned as the routine value.
3347 3460 1
3348 3461 1
3349 3462 2 BEGIN
3350 3463 2
3351 3464 2 MAP
3352 3465 2     REGDESCR: DBG$REGDESCR;           ! Input Register Descriptor
3353 3466 2
3354 3467 2 LOCAL
3355 3468 2     INDEX,                          ! Character index within REGTABLE entry
3356 3469 2     INVOCNUM,                       ! Invocation number for scope routine
3357 3470 2     LENGTH,                         ! Current length of ASCII string
3358 3471 2     MODRSTPTR,                     ! Module SYMID containing the scope
3359 3472 2     NAMEPTR: REF VECTOR[.BYTE],    ! Pointer to Counted ASCII string con-
3360 3473 2                                     ! taining the built register name
3361 3474 2     PATHDESC,                     ! Pointer to Pathname Descriptor for
3362 3475 2                                     ! routine name of desired scope
3363 3476 2     PATHSTRING: REF VECTOR[.BYTE], ! Pointer to Counted ASCII string for
3364 3477 2                                     ! routine name of desired scope
3365 3478 2     REGPTR: REF VECTOR[.BYTE],      ! Pointer to register name in REGTABLE
3366 3479 2     RSTPTR,                         ! Pointer to RST entry of scope routine
3367 3480 2     SCOPENUM,                      ! Temporary value of scope number
3368 3481 2     TEMPNUM,                       ! Temporary in computing scope number
3369 3482 2     TEMPSTR: VECTOR[12,BYTE];      ! Temporary buffer for scope number
3370 3483 2
3371 3484 2 BIND
3372 3485 2     REGTABLE = UPLIT ('R0 ', 'R1 ', 'R2 ', 'R3 ', 'R4 ', 'R5 ', 'R6 ', 'R7 ', 'R8 ', 'R9 ', 'R10 ', 'R11 ', 'AP ', 'FP ', 'SP ', 'PC ', 'PSL '); VECTOR[.LONG];
3373 3486 2
3374 3487 2
3375 3488 2
3376 3489 2
3377 3490 2
3378 3491 2
3379 3492 2
3380 3493 2     ! Check that we really have a valid Register Descriptor.
3381 3494 2
3382 3495 2     IF (.REGDESCR[DBG$V_REGD_SENTINEL] NEQ XX'2D') OR
```

```

: 3383      3496      3      (.REGDESCR[DBG$B_REGD_REGNUM] GTR 16)
: 3384      3497      2      THEN
: 3385      3498      2          $DBG_ERROR('RSTACCESS\REGISTER_NAME');
: 3386      3499      2
: 3387      3500      2
: 3388      3501      2      ! If the scope specified by the the Register Descriptor is in a set module,
: 3389      3502      2      ! we should be able to symbolize the scope as a routine name. We thus call
: 3390      3503      2      ! NUMBERED_SCOPE to get a routine SYMID for the scope. If this succeeds,
: 3391      3504      2      ! we convert that SYMID to a name string (in Counted ASCII) and leave that
: 3392      3505      2      ! name in a temporary memory block pointed to by NAMEPTR.
: 3393      3506      2
: 3394      3507      2      SCOPENUM = .REGDESCR[DBG$W_REGD_SCOPENUM];
: 3395      3508      2      IF .DBG$GB_MOD_PTR[MODE_SYMBOLS]
: 3396      3509      2      THEN
: 3397      3510      2          DBG$STA_NUMBERED_SCOPE(.SCOPENUM, MODRSTPTR, RSTPTR, INVOCNUM)
: 3398      3511      2      ELSE
: 3399      3512      2          MODRSTPTR = 0;
: 3400      3513      2
: 3401      3514      2      IF .MODRSTPTR NEQ 0
: 3402      3515      2      THEN
: 3403      3516      3          BEGIN
: 3404      3517      3              IF .INVOCNUM NEQ 0 THEN RSTPTR = DBG$BUILD_INVOC_RST(.RSTPTR, .INVOCNUM);
: 3405      3518      3              DBG$STA_SYMPATHNAME(.RSTPTR, PATHDESC);
: 3406      3519      3              DBG$NPATHDESC TO CS(.PATHDESC, PATHSTRING);
: 3407      3520      3              LENGTH = .PATHSTRING[0];
: 3408      3521      3              NAMEPTR = DBG$GET_TEMPMEM((.LENGTH + 8 + %UPVAL - 1)/%UPVAL);
: 3409      3522      3              CH$MOVE(.LENGTH + 1, .PATHSTRING, .NAMEPTR);
: 3410      3523      3              END
: 3411      3524      3
: 3412      3525      3
: 3413      3526      3      ! No specific routine name can be found for this scope in the RST. We
: 3414      3527      3      ! therefore must represent the scope by a scope number. We first get a
: 3415      3528      3      ! temporary memory block and initialize an empty Counted ASCII string in
: 3416      3529      3      ! it. We then convert the register's scope number to a decimal Counted
: 3417      3530      3      ! ASCII string. This becomes the numeric scope which precedes the actual
: 3418      3531      3      ! register name (for example, the '2' in '2\%R5').
: 3419      3532      3
: 3420      3533      2      ELSE
: 3421      3534      3          BEGIN
: 3422      3535      3              NAMEPTR = DBG$GET_TEMPMEM(5);
: 3423      3536      3              LENGTH = 0;
: 3424      3537      3              WHILE TRUE DO
: 3425      3538      4                  BEGIN
: 3426      3539      4                      TEMPNUM = .SCOPENUM/10;
: 3427      3540      4                      TEMPSTR[.LENGTH] = .SCOPENUM - .TEMPNUM*10 + '0';
: 3428      3541      4                      LENGTH = .LENGTH + 1;
: 3429      3542      4                      IF .TEMPNUM EQL 0 THEN EXITLOOP;
: 3430      3543      4                      SCOPENUM = .TEMPNUM;
: 3431      3544      4                      END;
: 3432      3545      3
: 3433      3546      3              INCR I FROM 1 TO .LENGTH DO
: 3434      3547      3                  NAMEPTR[I] = .TEMPSTR[.LENGTH - .I];
: 3435      3548      3
: 3436      3549      2          END;
: 3437      3550      2
: 3438      3551      2
: 3439      3552      2      ! We now have the scope name as either a routine name or a scope number in
```

```

: 3440      3553 2      ! the NAMEPTR buffer. Next we fill in the '\x' that separates the scope
: 3441      3554 2      ! name from the register name.
: 3442      3555 2
: 3443      3556 2      NAMEPTR[.LENGTH + 1] = '\x';
: 3444      3557 2      NAMEPTR[.LENGTH + 2] = '\x';
: 3445      3558 2      LENGTH = .LENGTH + 2;
: 3446      3559 2
: 3447      3560 2
: 3448      3561 2      ! Now fill in the register name itself. We look it up in REGTABLE, where
: 3449      3562 2      ! each register name is given as four ASCII characters. This code assumes
: 3450      3563 2      ! that each register name in REGTABLES ends with at least one blank.
: 3451      3564 2
: 3452      3565 2      REGPTR = REGTABLE[.REGDESCR[DBG$B_REGD_REGNUM]];
: 3453      3566 2      INDEX = 0;
: 3454      3567 2      WHILE .REGPTR[.INDEX] NEQ ' ' DO
: 3455      3568 2          BEGIN
: 3456      3569 3              LENGTH = .LENGTH + 1;
: 3457      3570 3              NAMEPTR[.LENGTH] = .REGPTR[.INDEX];
: 3458      3571 3              INDEX = .INDEX + 1;
: 3459      3572 2          END;
: 3460      3573 2
: 3461      3574 2
: 3462      3575 2      ! Finally, fill in '+offset' if the offset is non-zero. (The offset can
: 3463      3576 2      ! only have values 1, 2, or 3 in this case.)
: 3464      3577 2
: 3465      3578 2      IF .REGDESCR[DBG$V_REGD_OFFSET] NEQ 0
: 3466      3579 2      THEN
: 3467      3580 3          BEGIN
: 3468      3581 3              NAMEPTR[.LENGTH + 1] = '+';
: 3469      3582 3              NAMEPTR[.LENGTH + 2] = .REGDESCR[DBG$V_REGD_OFFSET] + '0';
: 3470      3583 3              LENGTH = .LENGTH + 2;
: 3471      3584 2          END;
: 3472      3585 2
: 3473      3586 2
: 3474      3587 2      ! Now return a pointer to the Counted ASCII register name string.
: 3475      3588 2
: 3476      3589 2      NAMEPTR[0] = .LENGTH;
: 3477      3590 2      RETURN .NAMEPTR;
: 3478      3591 2
: 3479      3592 1      END;
```

```

                                .PSECT  DBG$PLIT, NOWRT,  SHR,  PIC, 0
20 20 30 52 00168 P.AAX:  .ASCII  \R0  \
20 20 31 52 0016C      .ASCII  \R1  \
20 20 32 52 00170      .ASCII  \R2  \
20 20 33 52 00174      .ASCII  \R3  \
20 20 34 52 00178      .ASCII  \R4  \
20 20 35 52 0017C      .ASCII  \R5  \
20 20 36 52 00180      .ASCII  \R6  \
20 20 37 52 00184      .ASCII  \R7  \
20 20 38 52 00188      .ASCII  \R8  \
20 20 39 52 0018C      .ASCII  \R9  \
20 30 31 52 00190      .ASCII  \R10 \
20 31 31 52 00194      .ASCII  \R11 \
```

49	47	45	52	5C	53	53	45	43	43	41	20	20	50	41	00198	.ASCII	\AP	\
											20	20	50	46	0019C	.ASCII	\FP	\
											20	20	50	53	001A0	.ASCII	\SP	\
											20	20	43	50	001A4	.ASCII	\PC	\
											20	4C	53	50	001A8	.ASCII	\PSL	\
						45	4D	41	4E	5F	54	53	52	17	001AC	P.AAY: .ASCII	<23>\RSTACCESS\<92>\REGISTER_NAME\	
											52	45	54	53	001BB			

P.AAX

2D	04	AC		01FC	00000	.PSECT	DBG\$CODE, NOWRT, SHR, PIC, 0	
			58	00000000G	00	9E	00002	.ENTRY
			5E		20	C2	00009	DBG\$STA_REGISTER_NAME, Save R2,R3,R4,R5,R6,-;
			06		02	ED	0000C	R7,R8
					06	12	00012	MOVAB
			10	05	AC	91	00014	DBG\$GET_TEMP MEM, R8
					15	1B	00018	SUBL2
				00000000'	EF	9F	0001A	#32, SP-
					01	DD	00020	CMPZV
				00028362	8F	DD	00022	#2, #6, REGDESCR, #45
			00		03	FB	00028	BNEQ
			52	06	AC	3C	0002F	1\$
			50	00000000G	00	D0	00033	CMPB
			11		A0	E9	0003A	REGDESCR+1, #16
					5E	DD	0003E	2\$
					08	AE	00040	PUSHAB
					10	AE	00043	P.AAY
					52	DD	00046	#1
			FD16	CF	04	FB	00048	PUSHL
					03	11	0004D	#164706
					08	AE	0004F	CALLS
					08	AE	00052	#3, LIB\$SIGNAL
					47	13	00055	REGDESCR+2, SCOPENUM
					6E	D5	00057	DBG\$GB_MOD_PTR, R0
					0E	13	00059	2(R0), -3\$
					6E	DD	0005B	SP
					08	AE	0005D	PUSHL
			EA26	CF	02	FB	00060	PUSHAB
			04	AE	50	D0	00065	RSTPTR
					0C	AE	00069	MODRSTPTR
					08	AE	0006C	PUSHL
			0000V	CF	02	FB	0006F	SCOPENUM
					10	AE	00074	CALLS
					10	AE	00077	#4, DBG\$STA_NUMBERED_SCOPE
					02	FB	0007A	4\$
					10	BE	00081	CLRL
					0B	A6	00085	TSTL
					04	C7	00089	MODRSTPTR
					01	FB	0008D	6\$
					50	D0	00090	TSTL
					01	A6	00093	INVOCNUM
					50	28	00097	5\$
					38	11	0009C	INVOCNUM
								PUSHL
								RSTPTR
								CALLS
								#2, DBG\$BUILD_INVOC_RST
								MOV
								R0, RSTPTR
								PUSHAB
								PATHDESC
								PUSHL
								RSTPTR
								CALLS
								#2, DBG\$STA_SYMPATHNAME
								PUSHAB
								PATHSTRING
								PUSHL
								PATHDESC
								CALLS
								#2, DBG\$NPATHDESC_TO_CS

			05	DD	0009E	6\$:	PUSHL	#5		3535
	68		01	FB	000A0		CALLS	#1, DBG\$GET_TEMPNUM		
	57		50	D0	000A3		MOVL	R0, NAMEPTR		
			56	D4	000A6		CLRL	LENGTH		3536
50	52		0A	C7	000A8	7\$:	DIVL3	#10, SCOPENUM, TEMPNUM		3539
51	50		0A	C5	000AC		MULL3	#10, TEMPNUM, R1		3540
	51		52	C2	000B0		SUBL2	SCOPENUM, R1		
14 AE46	30		51	83	000B3		SUBB3	R1, #48, TEMPSTR[LENGTH]		
			56	D6	000B9		INCL	LENGTH		3541
			50	D5	000BB		TSTL	TEMPNUM		3542
	52		05	13	000BD		BEQL	8\$		
			50	D0	000BF		MOVL	TEMPNUM, SCOPENUM		3543
			E4	11	000C2		BRB	7\$		3537
			51	D4	000C4	8\$:	CLRL	1		3547
			0A	11	000C6		BRB	10\$		
50	56		51	C3	000C8	9\$:	SUBL3	1, LENGTH, R0		
	6147	14 AE40	90	000CC			MOVB	TEMPSTR[R0], (1)[NAMEPTR]		
F2	51		56	F3	000D2	10\$:	AOBLEQ	LENGTH, 1, 6\$		
	01 A647	5C	BF	90	000D6	11\$:	MOVB	#92, 1(LENGTH)[NAMEPTR]		3556
	02 A647		25	90	000DC		MOVB	#37, 2(LENGTH)[NAMEPTR]		3557
	56		02	C0	000E1		ADDL2	#2, LENGTH		3558
	50	05	AC	9A	000E4		MOVZBL	REGDESCR+1, R0		3565
	50	00000000	EF40	DE	000E8		MOVAL	REGTABLE[R0], REGPTR		
			51	D4	000F0		CLRL	INDEX		3566
	20		6140	91	000F2	12\$:	CMPB	(INDEX)[REGPTR], #32		3567
			09	13	000F6		BEQL	13\$		
			56	D6	000F8		INCL	LENGTH		3569
	6647		8140	90	000FA		MOVB	(INDEX)+[REGPTR], (LENGTH)[NAMEPTR]		3570
			F1	11	000FF		BRB	12\$		3567
	03	04	AC	93	00101	13\$:	BITB	REGDESCR, #3		3578
			14	13	00105		BEQL	14\$		
	01 A647		2B	90	00107		MOVB	#43, 1(LENGTH)[NAMEPTR]		3581
50	02		00	EF	0010C		EXTZV	#0, #2, REGDESCR, R0		3582
	50		30	81	00112		ADDB3	#48, R0, 2(LENGTH)[NAMEPTR]		
	56		02	C0	00118		ADDL2	#2, LENGTH		3583
	67		56	90	0011B	14\$:	MOVB	LENGTH, (NAMEPTR)		3589
	50		57	D0	0011E		MOVL	NAMEPTR, R0		3590
			04	00121			RET			3592

; Routine Size: 290 bytes, Routine Base: DBG\$CODE + 164E

```
3481 3593 1 GLOBAL ROUTINE DBG$STA_SAME_DST_OBJECT(SYMID1, SYMID2) =
3482 3594 1
3483 3595 1 FUNCTION
3484 3596 1     This routine determines whether two SYMIDs refer to the same DST object.
3485 3597 1     To do so, it checks that the corresponding RST entries have the same
3486 3598 1     kind and the same DST pointer. (Data records and their types point to
3487 3599 1     the same DST record; hence the kind must be checked as well.)
3488 3600 1
3489 3601 1 INPUTS
3490 3602 1     SYMID1 - The SYMID of the first of the two symbols to be compared.
3491 3603 1
3492 3604 1     SYMID2 - The SYMID of the second of the two symbols to be compared.
3493 3605 1
3494 3606 1 OUTPUTS
3495 3607 1     The value TRUE is returned if both symbols are of the same kind and have
3496 3608 1     the same SYMID. The value FALSE is returned otherwise.
3497 3609 1
3498 3610 1
3499 3611 2 BEGIN
3500 3612 2
3501 3613 2 MAP
3502 3614 2     SYMID1: REF RST$ENTRY,      ! Pointer to first RST entry
3503 3615 2     SYMID2: REF RST$ENTRY;    ! Pointer to second RST entry
3504 3616 2
3505 3617 2
3506 3618 2
3507 3619 2     ! Return the desired boolean value.
3508 3620 2
3509 3621 2 IF (.SYMID1[RST$B_KIND] EQL .SYMID2[RST$B_KIND]) AND
3510 3622 3     (.SYMID1[RST$L_DSTPTR] EQL .SYMID2[RST$L_DSTPTR])
3511 3623 2 THEN
3512 3624 2     RETURN TRUE;
3513 3625 2
3514 3626 2 RETURN FALSE;
3515 3627 2
3516 3628 1 END;
```

				0000 00000	.ENTRY	DBG\$STA_SAME_DST_OBJECT, Save nothing	3593
	51	04	AC	D0 00002	MOVL	SYMID1, R1	3621
	50	08	AC	D0 00006	MOVL	SYMID2, R0	
14	A0	14	A1	91 0000A	CMPB	20(R1), 20(R0)	
			0B	12 0000F	BNEQ	1\$	
0C	A0	0C	A1	D1 00011	CMPL	12(R1), 12(R0)	3622
			04	12 00016	BNEQ	1\$	
	50		01	D0 00018	MOVL	#1, R0	3624
				04 0001B	RET		
			50	D4 0001C 1\$:	CLRL	R0	3626
				04 0001E	RET		3628

; Routine Size: 31 bytes, Routine Base: DBG\$CODE + 1770

```
3518 3629 1 GLOBAL ROUTINE DBG$STA_SETCONTEXT(SYMID): NOVALUE =
3519 3630 1
3520 3631 1 FUNCTION
3521 3632 1 This routine sets up the context needed for subsequent DST value spec
3522 3633 1 evaluations. This specifically means determining the VAX CALL frame
3523 3634 1 and associated register values which are to be used for evaluating value
3524 3635 1 specs and determining symbol addresses. This routine must therefore
3525 3636 1 be called before routines DBG$STA_SYMTYPE, DBG$STA_SYMVALUE, and all
3526 3637 1 routines of the form DBG$STA_TYPE_XXX. Failure to do so may cause in-
3527 3638 1 correct value computations.
3528 3639 1
3529 3640 1 The context is defined by an input SYMID. The innermost invocable enti-
3530 3641 1 ty (i.e. routine) in the environment of the symbol's declaration is
3531 3642 1 looked up in the VAX CALL stack and the associated register set is loc-
3532 3643 1 ated. If an invocation number is attached to the SYMID, that is taken
3533 3644 1 into account. Context will not be established (and the previous context
3534 3645 1 will be deleted) if the input SYMID is zero or if the symbol's environ-
3535 3646 1 ment is not presently active. If context is not established, subsequent
3536 3647 1 value specs may still be evaluated, but if they refer to any register
3537 3648 1 values or locations (i.e., to any context) an error will be signalled.
3538 3649 1
3539 3650 1 INPUTS
3540 3651 1 SYMID - The SYMID of the symbol whose environment of declaration is to
3541 3652 1 be used to define the context of subsequent value spec. SYMID
3542 3653 1 must be of kind RST$K_DATA or RST$K_TYPCOMP. If SYMID is zero
3543 3654 1 no context is established.
3544 3655 1
3545 3656 1 OUTPUTS
3546 3657 1 NONE
3547 3658 1
3548 3659 1
3549 3660 2 BEGIN
3550 3661 2
3551 3662 2 MAP
3552 3663 2 SYMID: REF RST$ENTRY; : Pointer to the input RST entry
3553 3664 2
3554 3665 2 OWN
3555 3666 2 SPVALUE: REF VECTOR[.LONG]; : Current CALL frame's SP value
3556 3667 2
3557 3668 2 LOCAL
3558 3669 2 CURRENT_REG: REF VECTOR[.LONG], : Pointer to vector of current register
3559 3670 2 values (at top of stack)
3560 3671 2 ENDADDR, : The routine's PC end address
3561 3672 2 FRAME_FOUND_FLAG, : Flag set to TRUE when a CALL frame for
3562 3673 2 the desired routine is found
3563 3674 2 FRAMEPTR: REF BLOCK[.BYTE], : Pointer to current VAX CALL frame
3564 3675 2 INVOC_COUNT, : Number of invocations of routine
3565 3676 2 found so far in CALL stack
3566 3677 2 INVOCNUM, : The desired invocation number
3567 3678 2 INVPTR: REF RST$ENTRY, : Pointer to Invocation Number RST Entry
3568 3679 2 J, : CALL frame register-vector index
3569 3680 2 P[VAL, : Current CALL frame's PC value
3570 3681 2 MODPTR: REF RST$ENTRY, : Current Module
3571 3682 2 REGMASK: BITVECTOR[16], : Register save mask from the CALL frame
3572 3683 2 REGPTR: REF VECTOR[.LONG], : Pointer to a register's save location
3573 3684 2 REGSAVELOC: REF VECTOR[.LONG], : Pointer to CALL frame register save
3574 3685 2 area for registers R0 - R11
```

```
3575 3686 2 REGVEC: VECTOR[17, LONG],      . Vector of pointers to save areas for
3576 3687 2                               . the current frame's registers
3577 3688 2 ROUTPTR: REF RST$ENTRY,      . Pointer to Routine RST Entry of routine
3578 3689 2                               . to look for in CALL stack
3579 3690 2 RSTPTR: REF RST$ENTRY,      . RST pointer from SAT entry
3580 3691 2 RUNFRAME_PTR,           . Pointer to current entry in CALL command
3581 3692 2                               . runframe stack (needed by
3582 3693 2                               . the GET_REGISTER_VALUES routine)
3583 3694 2 SATPTR: REF SAT$ENTRY,      . Pointer to Static Address Table entry
3584 3695 2                               . for possible nested routine
3585 3696 2 SCOPE_IS_NUMERIC,          . Set to TRUE if context is a numeric
3586 3697 2                               . scope, not an RST routine entry
3587 3698 2 SCOPE_NUMBER,             . Scope number of the current context
3588 3699 2 STARTADDR;               . The routine's PC start address
3589 3700 2
3590 3701 2 ENABLE
3591 3702 2     SETCONTEXT_ERROR_HANDLER;      ! Set up error handler for this routine
3592 3703 2
3593 3704 2
3594 3705 2 ! If the input SYMID is zero, clear the current context and return.
3595 3706 2 !
3596 3707 2 DBG$SCOPE_NUMBER = 0;
3597 3708 2 IF .SYMID EQL 0
3598 3709 2 THEN
3599 3710 2     BEGIN
3600 3711 2         CURRENT_REG = DBG$RUNFRAME[DBG$L_USER_REGS];
3601 3712 2         DBG$REG_SYMID = 0;
3602 3713 2         INCR I FROM 0 TO 16 DO
3603 3714 2             BEGIN
3604 3715 2                 DBG$REG_VECTOR[I] = 0;
3605 3716 2                 DBG$REG_VALUES[I] = .CURRENT_REG[I];
3606 3717 2             END;
3607 3718 2
3608 3719 2     RETURN;
3609 3720 2     END;
3610 3721 2
3611 3722 2
3612 3723 2 ! We have a non-zero SYMID. Make sure SYMID is of a valid kind.
3613 3724 2 !
3614 3725 2 IF (.SYMID[RST$B_KIND] EQL RST$K_TYPE) OR
3615 3726 2     (.SYMID[RST$B_KIND] EQL RST$K_OVERLOAD)
3616 3727 2 THEN
3617 3728 2     $DBG_ERROR('RSTACCESS\SETCONTEXT');
3618 3729 2
3619 3730 2
3620 3731 2 ! Set the current context to 'not established'. We do this by zeroing all
3621 3732 2 ! the register save location pointers in DBG$REG_VECTOR. Also save the input
3622 3733 2 ! SYMID for later use in error messages.
3623 3734 2 !
3624 3735 2 INCR I FROM 0 TO 16 DO DBG$REG_VECTOR[I] = 0;
3625 3736 2 DBG$REG_SYMID = .SYMID;
3626 3737 2
3627 3738 2
3628 3739 2 ! Find the RST entry of the inner-most routine containing the declaration
3629 3740 2 ! of the SYMID symbol. If no such routine exists (because the symbol is
3630 3741 2 ! declared at the module level) return with no context set. If the context
3631 3742 2 ! is a numeric scope (i.e., the context N levels down in the VAX call
```

```
: 3632      3743 2      ! stack), we simply set the SCOPE_IS_NUMERIC flag and pick up the value of
: 3633      3744 2      ! N from the module RST entry--this is how the context is set for register
: 3634      3745 2      ! symbols in a specified numeric scope.
: 3635      3746 2
: 3636      3747 2      SCOPE_IS_NUMERIC = FALSE;
: 3637      3748 2      ROUTPTR = .SYMID;
: 3638      3749 2      WHILE .ROUTPTR[RST$B_KIND] NEQ RST$K_ROUTINE DO
: 3639      3750 3          BEGIN
: 3640      3751 3              IF .ROUTPTR[RST$B_KIND] EQL RST$K_MODULE
: 3641      3752 3                  THEN
: 3642      3753 4                      BEGIN
: 3643      3754 4                          IF NOT .ROUTPTR[RST$V_MODNUMSCP] THEN RETURN;
: 3644      3755 4                          SCOPE_IS_NUMERIC = TRUE;
: 3645      3756 4                          INVOCNUM = .ROUTPTR[RST$L_MODSCPNUM];
: 3646      3757 4                          STARTADDR = 0;
: 3647      3758 4                          ENDADDR = %X'FFFFFFFF';
: 3648      3759 4                          EXITLOOP;
: 3649      3760 3                      END;
: 3650      3761 3
: 3651      3762 3          ROUTPTR = .ROUTPTR[RST$L_UPSCOPEPTR];
: 3652      3763 2          END;
: 3653      3764 2
: 3654      3765 2
: 3655      3766 2      ! If this is a regular routine scope (i.e., it is not a numeric scope for
: 3656      3767 2      ! a register reference), pick up the routine PC address range and determine
: 3657      3768 2      ! what the corresponding invocation number is (default is zero).
: 3658      3769 2
: 3659      3770 2      IF NOT .SCOPE_IS_NUMERIC
: 3660      3771 2      THEN
: 3661      3772 3          BEGIN
: 3662      3773 3              STARTADDR = .ROUTPTR[RST$L_STARTADDR];
: 3663      3774 3              ENDADDR = .ROUTPTR[RST$L_ENDADDR];
: 3664      3775 3              INVOCNUM = 0;
: 3665      3776 3              IF .SYMID[RST$V_INVOCNUM]
: 3666      3777 3                  THEN
: 3667      3778 4                      BEGIN
: 3668      3779 4                          INVPTR = .SYMID[RST$L_SYMCHNPTR];
: 3669      3780 4                          INVOCNUM = .INVPTR[RST$L_INVOCNUM];
: 3670      3781 3                      END;
: 3671      3782 3
: 3672      3783 2          END;
: 3673      3784 2
: 3674      3785 2
: 3675      3786 2      ! Initialize the PC, the Frame Pointer, the scope number, and the register
: 3676      3787 2      ! values to their current (top of stack) values.
: 3677      3788 2
: 3678      3789 2      PCVAL = .DBG$RUNFRAME[DBG$L_USER_PC];
: 3679      3790 2      FRAMEPTR = .DBG$RUNFRAME[DBG$L_USER_FP];
: 3680      3791 2      SCOPE_NUMBER = 0;
: 3681      3792 2      CURRENT_REG = DBG$RUNFRAME[DBG$L_USER_REGS];
: 3682      3793 2      INCR I FROM 0 TO 16 DO
: 3683      3794 2          REGVEC[I] = CURRENT_REG[I];
: 3684      3795 2
: 3685      3796 2
: 3686      3797 2      ! Now search through the CALL frames on the VAX stack looking for the prop-
: 3687      3798 2      ! er invocation of the ROUTPTR routine or for the specified numeric scope.
: 3688      3799 2      ! Pick up all register save addresses in the stack along the way.
```

```
3689 3800 2
3690 3801 2
3691 3802 2
3692 3803 2
3693 3804 2
3694 3805 2
3695 3806 2
3696 3807 2
3697 3808 2
3698 3809 2
3699 3810 4
3700 3811 2
3701 3812 2
3702 3813 2
3703 3814 2
3704 3815 2
3705 3816 2
3706 3817 2
3707 3818 2
3708 3819 4
3709 3820 2
3710 3821 4
3711 3822 4
3712 3823 4
3713 3824 4
3714 3825 4
3715 3826 4
3716 3827 4
3717 3828 4
3718 3829 4
3719 3830 4
3720 3831 4
3721 3832 4
3722 3833 4
3723 3834 4
3724 3835 5
3725 3836 5
3726 3837 5
3727 3838 5
3728 3839 5
3729 3840 5
3730 3841 5
3731 3842 5
3732 3843 5
3733 3844 5
3734 3845 5
3735 3846 5
3736 3847 5
3737 3848 5
3738 3849 5
3739 3850 5
3740 3851 5
3741 3852 5
3742 3853 5
3743 3854 5
3744 3855 5
3745 3856 4
```

```
!
RUNFRAME_PTR = .DBG$RUNFRAME[DBG$NEXT_LINK];
INVOC_COUNT = 0;
WHILE TRUE DO
  BEGIN

    ! If we got to the bottom of the stack without finding the desired
    ! invocation, return with the context not set.
    IF (.PCVAL EQL 0) OR (.FRAMEPTR[SFS$HANDLER] EQL DBG$FINAL_HANDL)
    THEN
      RETURN;

    ! If this is a CALL frame of the routine we are looking for, increment
    ! the invocation count. When that reaches the desired invocation number
    ! we have found the desired CALL frame and exit the loop.
    IF (.PCVAL GEQU .STARTADDR) AND (.PCVAL LEQU .ENDADDR)
    THEN
      BEGIN

        ! The PC from this CALL frame is in the address range of the routine
        ! we are looking for. However, to make sure the PC is not really in
        ! a nested routine within the desired routine, we search the Module
        ! SAT starting at the desired routine's SAT entry looking for nested
        ! routines which cover the CALL frame's PC value. If we find such a
        ! routine, the CALL frame is not for the desired routine.

        FRAME_FOUND_FLAG = TRUE;
        SATPTR = 0;
        IF NOT .SCOPE_IS_NUMERIC
        THEN
          BEGIN
            SATPTR = .ROUTPTR[RST$L_RTNSATPTR];

            ! WARNING -- We can get into trouble here. Previously, we have
            ! assumed that the SAT is always around. This may not be the
            ! case if this module has been canceled. There are times when
            ! the module could be canceled and then set again to make us
            ! believe the the SAT is valid for this RST, but it is not! To
            ! correct the problem, when a module is canceled the field
            ! RST$L_RTNSATPTR is set to ZERO for each routine.
            ! So if the module for this RST has been canceled, SATPTR will
            ! be zero from the above statement. The problem is that this
            ! assumes there are no nested routines that truly require the
            ! correct context information. This is, of course, WRONG. A
            ! way of saving and getting to the SAT information must be
            ! found in the future. B.A. Becker MAY-1984

            IF .SATPTR NEQ 0
            THEN
              SATPTR = .SATPTR[SAT$L_FLINK];

          END;

        END;
```

```

: 3746      3857 4
: 3747      3858 4
: 3748      3859 5
: 3749      3860 5
: 3750      3861 5
: 3751      3862 5
: 3752      3863 5
: 3753      3864 5
: 3754      3865 5
: 3755      3866 5
: 3756      3867 5
: 3757      3868 5
: 3758      3869 5
: 3759      3870 5
: 3760      3871 5
: 3761      3872 5
: 3762      3873 5
: 3763      3874 5
: 3764      3875 5
: 3765      3876 6
: 3766      3877 5
: 3767      3878 6
: 3768      3879 6
: 3769      3880 6
: 3770      3881 5
: 3771      3882 5
: 3772      3883 5
: 3773      3884 5
: 3774      3885 5
: 3775      3886 5
: 3776      3887 4
: 3777      3888 4
: 3778      3889 4
: 3779      3890 4
: 3780      3891 4
: 3781      3892 4
: 3782      3893 4
: 3783      3894 4
: 3784      3895 4
: 3785      3896 4
: 3786      3897 5
: 3787      3898 5
: 3788      3899 5
: 3789      3900 4
: 3790      3901 4
: 3791      3902 3
: 3792      3903 3
: 3793      3904 3
: 3794      3905 3
: 3795      3906 3
: 3796      3907 3
: 3797      3908 3
: 3798      3909 3
: 3799      3910 3
: 3800      3911 3
: 3801      3912 3
: 3802      3913 3

      WHILE TRUE DO
      BEGIN

          ! If there are no more SAT entries in the chain or if they no
          ! longer cover the PCVAL address, exit the SAT loop.
          !
          IF .SATPTR EQL 0 THEN EXITLOOP;
          IF .SATPTR[SAT$L_START] GTRU .PCVAL THEN EXITLOOP;

          ! If this SAT entry is for a routine which covers the PCVAL
          ! address, we clear FRAME_FOUND_FLAG because the PC is in this
          ! nested routine instead of the routine we are looking for.
          RSTPTR = .SATPTR[SAT$L_RSTPTR];
          IF (.PCVAL GEQU .SATPTR[SAT$L_START]) AND
              (.PCVAL LEQU .SATPTR[SAT$L_END]) AND
              (.RSTPTR[RST$B_KIND] EQL RST$K_ROUTINE)
          THEN
              BEGIN
                  FRAME_FOUND_FLAG = FALSE;
                  EXITLOOP;
              END;

          ! Link on to the next SAT entry.
          !
          SATPTR = .SATPTR[SAT$L_FLINK];
          END;

          ! If the CALL frame we found really is for the desired routine,
          ! check the invocation count. If this is the desired invocation,
          ! exit the CALL stack loop. Otherwise, increment the invocation
          ! count and keep looping.
          !
          IF .FRAME_FOUND_FLAG
          THEN
              BEGIN
                  IF .INVOC_COUNT EQL .INVOCNUM THEN EXITLOOP;
                  INVOC_COUNT = .INVOC_COUNT + 1;
              END;

          END;

          ! We have not found the desired frame yet. Dig out the register save
          ! locations in this CALL frame and save those addresses in REGVEC.
          GET_REGISTER_VALUES(.FRAMEPTR, RUNFRAME_PTR, REGVEC);

          ! Determine what the value of SP (the Stack Pointer) is for the current
          ! CALL frame and save that in the OWN variable SPVALUE. Then make the
          ! save-location pointer in REGVEC point to SPVALUE. (Since SP does not
```

```
3803 3914 3      ! have a true save-location, the OWN variable fakes one.)
3804 3915 3
3805 3916 3      REGPTR = .REGVEC[14];
3806 3917 3      SPVALUE = .REGPTR[0];
3807 3918 3      REGVEC[14] = SPVALUE;
3808 3919 3
3809 3920 3
3810 3921 3      ! Dig out the values of PC and FP for the current CALL frame. Then
3811 3922 3      ! increment the scope number and loop for the next stack frame.
3812 3923 3
3813 3924 3      REGPTR = .REGVEC[15];
3814 3925 3      PCVAL = .REGPTR[0];
3815 3926 3      REGPTR = .REGVEC[13];
3816 3927 3      FRAMEPTR = .REGPTR[0];
3817 3928 3      SCOPE_NUMBER = .SCOPE_NUMBER + 1;
3818 3929 3      END;
3819 3930 2
3820 3931 2
3821 3932 2      ! We have found the CALL frame and thus the context we wanted. Set the
3822 3933 2      ! address of each register's save location in DBG$REG_VECTOR and the regis-
3823 3934 2      ! ter's value in DBG$REG_VALUES. This makes the context available to the
3824 3935 2      ! value spec routines. Then set the scope number in DBG$SCOPE_NUMBER and
3825 3936 2      ! return to the caller.
3826 3937 2
3827 3938 2      INCR I FROM 0 TO 16 DO
3828 3939 3      BEGIN
3829 3940 3      REGPTR = .REGVEC[.I];
3830 3941 3      DBG$REG_VECTOR[.I] = .REGPTR;
3831 3942 3      IF .REGPTR NEQ 0 THEN DBG$REG_VALUES[.I] = .REGPTR[0];
3832 3943 3      END;
3833 3944 2
3834 3945 2      DBG$REG_VALUES[16] = (.DBG$REG_VALUES[16] AND %X'0000FFFF') OR
3835 3946 2      (.DBG$RUNFRAME[DBG$L_USER_PSL] AND %X'FFFF0000');
3836 3947 2      DBG$SCOPE_NUMBER = .SCOPE_NUMBER;
3837 3948 2      RETURN;
3838 3949 2
3839 3950 1      END;
```

```
43 54 45 53 5C 53 53 45 43 43 41 54 53 52 14 001C4 P.AAZ: .ASCII <20>\RSTACCESS\<92>\SETCONTEXT\
54 58 45 54 4E 4F 001D3
```

```
.PSECT DBG$OWN,NOEXE, PIC,2
00050 SPVALUE:.BLKB 4
```

```
.PSECT DBG$CODE,NOWRT, SHR, PIC,0
OFFC 00000
5E AC AE 9E 00002 .ENTRY DBG$STA SETCONTEXT, Save R2,R3,R4,R5,R6,R7,-: 3629
6D 01A8 CF DE 00006 MOVAB -84(SP), SP
MOVAL 22$, (FP) : 3660
```


		00000000'	EF	D4	00008	CLRL	DBG\$SCOPE_NUMBER	3707	
	52	04	AC	D0	00011	MOVL	SYMID, R2	3708	
			24	12	00015	BNEQ	2\$		
	54	00000000G	00	9E	00017	MOVAB	DBG\$RUNFRAME+4, CURRENT_REG	3711	
		00000000'	EF	D4	0001E	CLRL	DBG\$REG_SYMID	3712	
			50	D4	00024	CLRL	I	3716	
		00000000G0040	00	D4	00026	CLRL	DBG\$REG_VECTOR[I]	3715	
			6440	D0	0002D	MOVL	(CURRENT_REG)[I], DBG\$REG_VALUES[I]	3716	
EC	50		10	F3	00036	AOBLEQ	#16, I, T\$	3713	
				04	0003A	RET		3710	
	07	14	A2	91	0003B	2\$: CMPB	20(R2), #7	3725	
			06	13	0003F	BEQL	3\$		
	0D	14	A2	91	00041	CMPB	20(R2), #13	3726	
			15	12	00045	BNEQ	4\$		
		00000000'	EF	9F	00047	3\$: PUSHAB	P.AAZ	3728	
			01	DD	0004D	PUSHL	#1		
		00028362	8F	DD	0004F	PUSHL	#164706		
	00		03	FB	00055	CALLS	#3, LIB\$SIGNAL		
			50	D4	0005C	4\$: CLRL	I	3735	
		00000000G0040	00	D4	0005E	5\$: CLRL	DBG\$REG_VECTOR[I]		
F5	50		10	F3	00065	AOBLEQ	#16, I, 5\$		
		00000000'	EF	D0	00069	MOVL	R2, DBG\$REG_SYMID	3736	
			04	AE	D4	00070	CLRL	SCOPE IS NUMERIC	3747
	53		52	D0	00073	MOVL	R2, ROUTPTR	3748	
	02	14	A3	91	00076	6\$: CMPB	20(ROUTPTR), #2	3749	
			22	13	0007A	BEQL	9\$		
	01	14	A3	91	0007C	CMPB	20(ROUTPTR), #1	3751	
			16	12	00080	BNEQ	8\$		
01	28	A3	03	E0	00082	BBS	#3, 40(ROUTPTR), 7\$	3754	
				04	00087	RET			
	04	AE	01	D0	00088	7\$: MOVL	#1, SCOPE IS NUMERIC	3755	
		SA	20	A3	D0	0008C	MOVL	32(ROUTPTR), INVOCNUM	3756
			5B	D4	00090	CLRL	STARTADDR	3757	
	08	AE	01	CE	00092	MNEGL	#1, ENDADDR	3758	
			06	11	00096	BRB	9\$	3753	
	53	10	A3	D0	00098	8\$: MOVL	16(ROUTPTR), ROUTPTR	3762	
			D8	11	0009C	BRB	6\$	3749	
	18	04	AE	E8	0009E	9\$: BLBS	SCOPE IS NUMERIC, 10\$	3770	
		5B	18	A3	D0	000A2	MOVL	24(ROUTPTR), STARTADDR	3773
	08	AE	1C	A3	D0	000A6	MOVL	28(ROUTPTR), ENDADDR	3774
			5A	D4	000AB	CLRL	INVOCNUM	3775	
08	15	A2	02	E1	000AD	BBC	#2, 21(R2), 10\$	3776	
		50	08	A2	D0	000B2	MOVL	8(R2), INVPTR	3779
		5A	18	A0	D0	000B6	MOVL	24(INVPTR), INVOCNUM	3780
		55	00000000G	00	D0	000BA	10\$: MOVL	DBG\$RUNFRAME+64, PCVAL	3789
		57	00000000G	00	D0	000C1	MOVL	DBG\$RUNFRAME+56, FRAMEPTR	3790
			6E	D4	000C8	CLRL	SCOPE_NUMBER	3791	
	54	00000000G	00	9E	000CA	MOVAB	DBG\$RUNFRAME+4, CURRENT_REG	3792	
			50	D4	000D1	CLRL	I	3794	
	10	AE40	6440	DE	000D3	11\$: MOVL	(CURRENT_REG)[I], REGVEC[I]		
F6	50		10	F3	000D9	AOBLEQ	#16, I, T1\$		
	0C	AE	00000000G	00	D0	000DD	MOVL	DBG\$RUNFRAME, RUNFRAME_PTR	3801
			58	D4	000E5	CLRL	INVOC_COUNT	3802	
			55	D5	000E7	12\$: TSTL	PCVAL	3810	
			0A	13	000E9	BEQL	13\$		
	50	00000000G	00	9E	000EB	MOVAB	DBG\$FINAL_HANDL, R0		
	50		67	D1	000F2	CMPL	(FRAMEPTR), R0		

			01	12	000F5	13\$:	BNEQ	14\$		
				04	000F7		RET			
	5B		55	D1	000F8	14\$:	CMPL	PCVAL, STARTADDR		3819
			42	1F	000FB		BLSSU	18\$		
08	AE		55	D1	000FD		CMPL	PCVAL, ENDADDR		
			3C	1A	00101		BGTRU	18\$		
	59		01	D0	00103		MOVL	#1, FRAME_FOUND_FLAG		3831
			52	D4	00106		CLRL	SATPTR		3832
	09	04	AE	E8	00108		BLBS	SCOPE_IS_NUMERIC, 16\$		3833
	52	20	A3	D0	0010C		MOVL	32(ROOTPTR), SATPTR		3836
			23	13	00110		BEQL	17\$		3852
	52		62	D0	00112	15\$:	MOVL	(SATPTR), SATPTR		3854
			1E	13	00115	16\$:	BEQL	17\$		3865
	55	04	A2	D1	00117		CMPL	4(SATPTR), PCVAL		3866
			18	1A	0011B		BGTRU	17\$		
	56	0C	A2	D0	0011D		MOVL	12(SATPTR), RSTPTR		3873
04	A2		55	D1	00121		CMPL	PCVAL, 4(SATPTR)		3874
			EB	1F	00125		BLSSU	15\$		
08	A2		55	D1	00127		CMPL	PCVAL, 8(SATPTR)		3875
			E5	1A	0012B		BGTRU	15\$		
	02	14	A6	91	0012D		CMPB	20(RSTPTR), #2		3876
			DF	12	00131		BNEQ	15\$		
			59	D4	00133		CLRL	FRAME_FOUND_FLAG		3879
	07		59	E9	00135	17\$:	BLBC	FRAME_FOUND_FLAG, 18\$		3895
	5A		58	D1	00138		CMPL	INVOC_COUNT, INVOCNUM		3898
			35	13	0013B		BEQL	19\$		
			58	D6	0013D		INCL	INVOC_COUNT		3899
		10	AE	9F	0013F	18\$:	PUSHAB	REGVEC		3908
		10	AE	9F	00142		PUSHAB	RUNFRAME_PTR		
			57	DD	00145		PUSHL	FRAMEPTR		
0000V	CF		03	FB	00147		CALLS	#3, GET_REGISTER_VALUES		
	54	48	AE	D0	0014C		MOVL	REGVEC+56, REGPTR		3916
00000000'	EF		64	D0	00150		MOVL	(REGPTR), SPVALUE		3917
48	AE	00000000'	EF	9E	00157		MOVAB	SPVALUE, REGVEC+56		3918
	54	4C	AE	D0	0015F		MOVL	REGVEC+60, REGPTR		3924
	55		64	D0	00163		MOVL	(REGPTR), PCVAL		3925
	54	44	AE	D0	00166		MOVL	REGVEC+52, REGPTR		3926
	57		64	D0	0016A		MOVL	(REGPTR), FRAMEPTR		3927
			6E	D6	0016D		INCL	SCOPE_NUMBER		3928
		FF75	31	0016F			BRW	12\$		3803
			50	D4	00172	19\$:	CLRL	1		3938
	54	10	AE40	D0	00174	20\$:	MOVL	REGVEC[1], REGPTR		3940
00000000G0040			54	D0	00179		MOVL	REGPTR, DBG\$REG_VECTOR[1]		3941
			08	13	00181		BEQL	21\$		3942
00000000G0040			64	D0	00183		MOVL	(REGPTR), DBG\$REG_VALUES[1]		
E5	50	00000000G	00	F3	0018B	21\$:	AOBLEQ	#16, 1, 20\$		3938
			8F	CB	0018F		BICL3	#65535, DBG\$RUNFRAME+68, R0		3946
	51	00000000G	00	3C	0019B		MOVZWL	DBG\$REG_VALUES+64, R1		
00000000G	00		51	C9	001A2		BISL3	R1, R0, DBG\$REG_VALUES+64		
			6E	D0	001AA		MOVL	SCOPE_NUMBER, DBG\$SCOPE_NUMBER		3947
	00000000'		04	001B1			RET			3950
			0000	001B2	22\$:		.WORD	Save nothing		3660
			7E	D4	001B4		CLRL	-(SP)		
			5E	DD	001B6		PUSHL	SP		
	7E	04	AC	7D	001B8		MOVQ	4(AP), -(SP)		
0000V	CF		03	FB	001BC		CALLS	#3, SETCONTEXT_ERROR_HANDLER		
			04	001C1			RET			

RSTACK
V04-000

1 10
16-Sep-1984 02:48:17
14-Sep-1984 12:18:26

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]RSTACK.B32;1

Page 117
(23)

; Routine Size: 450 bytes, Routine Base: DBGSCODE + 178F

1

```
3841 3951 1 GLOBAL ROUTINE DBG$STA_SETREGISTERS: NOVALUE =
3842 3952 1
3843 3953 1 FUNCTION
3844 3954 1     This routine re-sets all register values in the current context (as
3845 3955 1     established by DBG$STA_SETCONTEXT) from the DBG$REG_VALUES vector.
3846 3956 1     This is done by copying each register's value from DBG$REG_VALUES to
3847 3957 1     the register save location in the VAX CALL stack (or in the Debugger's
3848 3958 1     save area for the top of stack register set). The addresses of these
3849 3959 1     save locations is given by DBG$REG_VECTOR. This routine must be called
3850 3960 1     at the end of each DEPOSIT command since this is the command which may
3851 3961 1     have changed the values of the registers in the current context.
3852 3962 1
3853 3963 1     As a side effect, this routine also clears the current context. It is
3854 3964 1     thus necessary to call DBG$STA_SETCONTEXT again before evaluating more
3855 3965 1     value specs containing register references.
3856 3966 1
3857 3967 1 INPUTS
3858 3968 1     DBG$REG_VECTOR and DBG$REG_VALUES are the implicit inputs. There are
3859 3969 1     no input parameters.
3860 3970 1
3861 3971 1 OUTPUTS
3862 3972 1     NONE
3863 3973 1
3864 3974 1
3865 3975 2 BEGIN
3866 3976 2
3867 3977 2 LOCAL
3868 3978 2     REGPTR: REF VECTOR[.LONG],      ! Pointer to register save location
3869 3979 2     PSWPTR: REF VECTOR[.WORD];      ! Pointer to PSW save location
3870 3980 2
3871 3981 2
3872 3982 2
3873 3983 2     ! Loop over the register set, re-setting all register values we can in the
3874 3984 2     ! current context. Note that SP (R14) cannot be explicitly restored.
3875 3985 2
3876 3986 2     DBG$REG_VECTOR[14] = 0;
3877 3987 2     INCR I FROM 0 TO 15 DO
3878 3988 3         BEGIN
3879 3989 3             REGPTR = .DBG$REG_VECTOR[.I];
3880 3990 3             IF .REGPTR NEQ 0 THEN REGPTR[0] = .DBG$REG_VALUES[.I];
3881 3991 3             DBG$REG_VECTOR[.I] = 0;
3882 3992 3             END;
3883 3993 2
3884 3994 2
3885 3995 2     ! Also re-set the Processor Status Word (PSW) in its save location.
3886 3996 2     ! Then return.
3887 3997 2
3888 3998 2     PSWPTR = .DBG$REG_VECTOR[16];
3889 3999 2     IF .PSWPTR NEQ 0 THEN PSWPTR[0] = .DBG$REG_VALUES[16];
3890 4000 2     DBG$REG_VECTOR[16] = 0;
3891 4001 2     RETURN;
3892 4002 2
3893 4003 1 END;
```

		000C	00000	.ENTRY	DBG\$STA_SETREGISTERS, Save R2,R3	:	3951
53	00000000G	00	9E 00002	MOVAB	DBG\$REG_VECTOR+64, R3	:	
	F8	A3	D4 00009	CLRL	DBG\$REG_VECTOR+56	:	3986
		50	D4 00C0C	CLRL	I	:	3987
51	CO A340	DE	0000E 1\$:	MOVAL	DBG\$REG_VECTOR[I], R1	:	3989
52		61	D0 00013	MOVL	(R1), REGPTR	:	
		08	13 00016	BEQL	2\$:	3990
62	00000000G0040	D0	00018	MOVL	DBG\$REG_VALUES[I], (REGPTR)	:	
		61	D4 00020 2\$:	CLRL	(R1)	:	3991
E8	50	0F	F3 00022	AOBLEQ	#15, I, 1\$:	3987
	50	63	D0 00026	MOVL	DBG\$REG_VECTOR+64, PSWPTR	:	3998
		07	13 00029	BEQL	3\$:	3999
60	00000000G	00	B0 0002B	MOVW	DBG\$REG_VALUES+64, (PSWPTR)	:	
		63	D4 00032 3\$:	CLRL	DBG\$REG_VECTOR+64	:	4000
		04	00034	RET		:	4003

; Routine Size: 53 bytes, Routine Base: DBG\$CODE + 1951

```
3895 4004 1 GLOBAL ROUTINE DBG$STA_SYM_IS_LITERAL (SYMID) =
3896 4005 1
3897 4006 1 FUNCTION
3898 4007 1     This routine accepts a symbol identifier and determines whether
3899 4008 1     the symbol represents a literal value. The same information can
3900 4009 1     be obtained by calling SYMVALUE, but that routine may have
3901 4010 1     side effects. This routine uses the same logic as SYMVALUE
3902 4011 1     and its subroutines VALSPEC and EVAL_MAT_SPEC, but does
3903 4012 1     not have the side effects associated with actually computing
3904 4013 1     the value.
3905 4014 1
3906 4015 1 INPUTS
3907 4016 1     SYMID - A longword symbol identifier previously produced by routine
3908 4017 1     DBG$STA_GETSYMBOL or DBG$STA_GETSYMOFF. SYMID uniquely ident-
3909 4018 1     ifies the symbol whose "value" is to be returned.
3910 4019 1
3911 4020 1 OUTPUTS
3912 4021 1     The return value is one of:
3913 4022 1     TRUE - The symbol does represent a literal
3914 4023 1     FALSE - The symbol does not represent a literal
3915 4024 1
3916 4025 1
3917 4026 2 BEGIN
3918 4027 2
3919 4028 2 MAP
3920 4029 2     SYMID: REF RST$ENTRY; ! Pointer to input symbol's RST entry
3921 4030 2
3922 4031 2 LOCAL
3923 4032 2     BLITRLR: REF DST$BLI_TRAILER1; ! Pointer to Bliss DST record trailer
3924 4033 2     BLIVALSPEC: BLOCK[8, BYTE] ! Value Spec buffer for Bliss special
3925 4034 2     FIELD(DST$VS_HDR_FIELDS); ! cases DST record
3926 4035 2     DSTPTR: REF DST$RECORD; ! Pointer to symbol's DST record
3927 4036 2     MSPTR: REF DST$MATER_SPEC; ! Pointer to a Materialization Spec
3928 4037 2     VSPTR: REF DST$VAL_SPEC; ! Pointer to a Value Spec
3929 4038 2
3930 4039 2
3931 4040 2
3932 4041 2 ! If the input symid is zero, return "false" for "does not represent
3933 4042 2 ! a literal". We need to check this here so we don't accvio later on
3934 4043 2 ! in the routine.
3935 4044 2
3936 4045 2 IF .SYMID EQL 0 THEN RETURN FALSE;
3937 4046 2
3938 4047 2
3939 4048 2 ! If the RST kind is not data, then the symbol is not a literal.
3940 4049 2
3941 4050 2 IF .SYMID[RST$B_KIND] NEQ RST$K_DATA
3942 4051 2 THEN
3943 4052 2     RETURN FALSE;
3944 4053 2
3945 4054 2
3946 4055 2 ! For RST records which are of kind data, obtain the DST record
3947 4056 2 ! which holds the value specification and act accordingly.
3948 4057 2
3949 4058 2 DSTPTR = .SYMID[RST$B_DSTPTR];
3950 4059 2 CASE .DSTPTR[DST$B_TYPE] FROM 0 TO 255 OF
3951 4060 2     SET
```

```

3952 4061 2
3953 4062 2
3954 4063 2
3955 4064 2
3956 4065 2
3957 4066 2
3958 4067 2
3959 4068 2
3960 4069 2
3961 4070 2
3962 4071 2
3963 4072 2
3964 4073 2
3965 4074 2
3966 4075 2
3967 4076 2
3968 4077 2
3969 4078 2
3970 4079 2
3971 4080 2
3972 4081 2
3973 4082 2
3974 4083 2
3975 4084 2
3976 4085 2
3977 4086 2
3978 4087 2
3979 4088 2
3980 4089 2
3981 4090 2
3982 4091 2
3983 4092 2
3984 4093 2
3985 4094 2
3986 4095 2
3987 4096 2
3988 4097 2
3989 4098 2
3990 4099 2
3991 4100 2
3992 4101 2
3993 4102 2
3994 4103 2
3995 4104 2
3996 4105 2
3997 4106 2
3998 4107 2
3999 4108 2
4000 4109 2
4001 4110 2
4002 4111 2
4003 4112 2
4004 4113 2
4005 4114 2
4006 4115 2
4007 4116 2
4008 4117 2

: Handle all normal DST records, i.e. those of the standard format.
: Obtain a pointer to the Value Spec.
[DST$K_DTYPE_LOWEST TO DST$K_DTYPE_HIGHEST,
DST$K_BOOL, DST$K_SEPTYP, DST$K_LBLORLIT,
DST$K_ENTRY, DST$K_RTNBEG, DST$K_BLKBEQ,
DST$K_RECBEQ, DST$K_ENUMELT]:
  VSPTR = DSTPTR[DST$B_VFLAGS];

: Handle the Bliss Special Cases DST record. Construct a Value Spec
: from the VFLAGS and VALUE fields in the record (which are not adjacent
: in this particular record).
[DST$K_BLI]:
  BEGIN
    BLIVALSPEC[DST$B_VS_VFLAGS] = .DSTPTR[DST$B_BLI_VFLAGS];
    BLITRLR = DSTPTR[DST$A_BLI_TRLR1] + .DSTPTR[DST$B_BLI_LNG];
    BLIVALSPEC[DST$L_VS_VALUE] = .BLITRLR[DST$L_BLI_VALUE];
    VSPTR = BLIVALSPEC;

: See the corresponding hack in DBG$STA_SYMVALUE.
IF .VSPTR[DST$V_VS_VALKIND] EQL DST$K_VALKIND_LITERAL
THEN
  VSPTR[DST$V_VS_VALKIND] = DST$K_VALKIND_ADDR;
END;

: BLISS fields. Return TRUE - these are literal values.
[DST$K_BLIFLD]:
  RETURN TRUE;

: Any other DST record does not represent a literal.
[INRANGE]:
  RETURN FALSE;

TES;

: If we fall through to here, VSPTR points to a Value Spec.
: If the value is given by a trailing Value Spec, we get to that Value
: Spec. We loop in case the indirection is repeated.
WHILE .VSPTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_TVS DO
  VSPTR = VSPTR[DST$A_VS_TVS_BASE] + .VSPTR[DST$L_VS_TVS_OFFSET];

: If the Value Spec gives the offset to a descriptor (in the DST),
: or the Value Spec is a Bit Offset Value Spec, then it does not
: represent a literal.
IF .VSPTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_DSC
```

```
4009 4118 2
4010 4119 2
4011 4120 2
4012 4121 2
4013 4122 2
4014 4123 2
4015 4124 2
4016 4125 2
4017 4126 2
4018 4127 2
4019 4128 2
4020 4129 2
4021 4130 2
4022 4131 2
4023 4132 2
4024 4133 2
4025 4134 2
4026 4135 2
4027 4136 2
4028 4137 2
4029 4138 2
4030 4139 2
4031 4140 2
4032 4141 2
4033 4142 2
4034 4143 2
4035 4144 2
4036 4145 2
4037 4146 2
4038 4147 2
4039 4148 2
4040 4149 2
4041 4150 2
4042 4151 2
4043 4152 2
4044 4153 2
4045 4154 2
4046 4155 2
4047 4156 2
4048 4157 2
4049 4158 2
4050 4159 2
4051 4160 2
4052 4161 2
4053 4162 2
4054 4163 2
4055 4164 1
```

```
OR .VSPTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_BITOFFS
THEN
  RETURN FALSE;

! If this is a Value-Spec-Follows value spec, a more complex value spec
! follows the VFLAGS field.
IF .VSPTR[DST$B_VS_VFLAGS] EQL DST$K_VS_FOLLOWS
THEN
  BEGIN

    ! If the object is not statically allocated, then it is not a literal.
    IF .VSPTR[DST$B_VS_ALLOC] NEQ DST$K_VS_ALLOC_STAT
    THEN
      RETURN FALSE;

    ! If we get here, the object is statically allocated.
    ! Obtain the Materialization Spec.
    MSPTR = VSPTR[DST$A_VS_MATSPEC];

    ! If the Materialization Spec is of kind 'R-Value', then
    ! it is a literal.
    IF .MSPTR[DST$B_MS_KIND] EQL DST$K_MS_RVAL
    THEN
      RETURN TRUE

    ELSE
      RETURN FALSE;
    END;

    ! If we fall through to here, we have an ordinary garden-variety
    ! Value Spec. If it is a literal, return true.
    IF .VSPTR[DST$V_VS_VALKIND] EQL DST$K_VALKIND_LITERAL
    THEN
      RETURN TRUE

    ELSE
      RETURN FALSE;
    END;
```

```
SE          0004 00000      .ENTRY  DBG$STA_SYM_IS_LITERAL, Save R2
50          08 C2 00002      SUBL2   #8, SP
          04 AC D0 00005      MOVL    SYMID, R0
          03 12 00009      BNEQ     2$
          026F 31 0000B 1$:      BRW    11$
```

```
: 4004
:
: 4045
:
```


		06	14	A0	91	0000E	2\$:	CMPB	20(R0), #6		4050	
		50		F7	12	00012		BNEQ	1\$			
		00	0C	A0	D0	00014		MOVL	12(R0), DSTPTR		4058	
	FF	8F	01	A0	8F	00018		CASEB	1(DSTPTR), #0, #255		4059	
0200	0200	0200		0206		0001E	3\$:	.WORD	5\$-3\$, -			
0200	0200	0200		0200		00026			4\$-3\$, -			
0200	0200	0200		0200		0002E			4\$-3\$, -			
0200	0200	0200		0200		00036			4\$-3\$, -			
0200	0200	0200		0200		0003E			4\$-3\$, -			
0200	0200	0200		0200		00046			4\$-3\$, -			
0200	0200	0200		0200		0004E			4\$-3\$, -			
0200	0200	0200		0200		00056			4\$-3\$, -			
0200	0200	0200		0200		0005E			4\$-3\$, -			
025F	025F	0200		0200		00066			4\$-3\$, -			
025F	025F	025F		025F		0006E			4\$-3\$, -			
025F	025F	025F		025F		00076			4\$-3\$, -			
025F	025F	025F		025F		0007E			4\$-3\$, -			
025F	025F	025F		025F		00086			4\$-3\$, -			
025F	025F	025F		025F		0008E			4\$-3\$, -			
025F	025F	025F		025F		00096			4\$-3\$, -			
025F	025F	025F		025F		0009E			4\$-3\$, -			
025F	025F	025F		025F		000A6			4\$-3\$, -			
025F	025F	025F		025F		000AE			4\$-3\$, -			
025F	025F	025F		025F		000B6			4\$-3\$, -			
025F	025F	025F		025F		000BE			4\$-3\$, -			
025F	025F	025F		025F		000C6			4\$-3\$, -			
025F	025F	025F		025F		000CE			4\$-3\$, -			
025F	025F	025F		025F		000D6			4\$-3\$, -			
025F	025F	025F		025F		000DE			4\$-3\$, -			
025F	025F	025F		025F		000E6			4\$-3\$, -			
025F	025F	025F		025F		000EE			4\$-3\$, -			
025F	025F	025F		025F		000F6			4\$-3\$, -			
025F	025F	025F		025F		000FE			4\$-3\$, -			
025F	025F	025F		025F		00106			4\$-3\$, -			
025F	025F	025F		025F		0010E			4\$-3\$, -			
025F	025F	025F		025F		00116			4\$-3\$, -			
025F	025F	025F		025F		0011E			4\$-3\$, -			
025F	025F	025F		025F		00126			4\$-3\$, -			
025F	025F	025F		025F		0012E			4\$-3\$, -			
025F	025F	025F		025F		00136			4\$-3\$, -			
025F	025F	025F		025F		0013E			4\$-3\$, -			
025F	025F	025F		025F		00146			4\$-3\$, -			
025F	025F	025F		025F		0014E			11\$-3\$, -			
025F	0200	025F		025F		00156			11\$-3\$, -			
0200	025F	025F		025F		0015E			11\$-3\$, -			
025F	025F	025F		0200		00166			11\$-3\$, -			
0200	025F	025F		025F		0016E			11\$-3\$, -			
025F	025F	025F		025F		00176			11\$-3\$, -			
025F	025F	025F		0200		0017E			11\$-3\$, -			
025B	025F	0200		025F		00186			11\$-3\$, -			
025F	0200	025F		025F		0018E			11\$-3\$, -			
025F	0200	025F		025F		00196			11\$-3\$, -			
025F	025F	025F		025F		0019E			11\$-3\$, -			
025F	025F	025F		025F		001A6			11\$-3\$, -			
025F	025F	025F		025F		001AE			11\$-3\$, -			
025F	025F	025F		025F		001B6			11\$-3\$, -			
025F	025F	025F		025F		001BE			11\$-3\$, -			

RSTACCESS
V04-000

```
C 11
16-Sep-1984 02:48:17      VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26      [DEBUG.SRC]RSTACCESS.B32;1
```

Page 124
(25)

025F
025F
025F
025F
025F
025F
025F
025F
025F
025F
025F

025F
025F
025F
025F
025F
025F
025F
025F
025F
025F
025F

025F
025F
025F
025F
025F
025F
025F
025F
025F
025F
025F

025F
025F
025F
025F
025F
025F
025F
025F
025F
025F
025F

001C6
001CE
001D6
001DE
001E6
001EE
001F6
001FE
00206
0020E
00216

[illegible]

.....

RSTACCESS
V04-000

```

D 11
16-Sep-1984 02:48:17      VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26      [DEBUG.SRC]RSTACCESS.B32;1

```

Page 125
(25)

RC
VC

RSTACCESS
V04-000

```
E 11
16-Sep-1984 02:48:17 YAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACKACCESS.B32;1
```

Page 126
(23)

RSTACCESS
V04-000

G 11
16-Sep-1984 02:48:17
14-Sep-1984 12:18:26

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]RSTACCESS.B32;1

Page 128
(25)

R
V

03	03	11	00272	BRB	98	:
	61	93	00274	BITB	(VSPTR), #3	: 4158
	04	12	00277	BNEQ	118	:
50	01	D0	00279	MOVL	#1, R0	: 4163
		04	0027C	RET		:
	50	D4	0027D	CLRL	R0	: 4164
		04	0027F	RET		:

; Routine Size: 640 bytes, Routine Base: DBG\$CODE + 1986

```
: 4057 4165 1 GLOBAL ROUTINE DBG$STA_SYMKIND(SYMID, KIND): NOVALUE =
: 4058 4166 1
: 4059 4167 1 FUNCTION
: 4060 4168 1     This routine returns the 'kind' of a specified symbol. The symbol is
: 4061 4169 1     represented by a symbol identifier as produced by DBG$STA_GETSYMBOL or
: 4062 4170 1     DBG$STA_GETSYMOCFF. The returned 'kind' is the same kind as originally
: 4063 4171 1     produced by those two routines.
: 4064 4172 1
: 4065 4173 1 INPUTS
: 4066 4174 1     SYMID - A longword symbol identifier previously produced by routine
: 4067 4175 1           DBG$STA_GETSYMBOL or DBG$STA_GETSYMOCFF. SYMID uniquely ident-
: 4068 4176 1           ifies the symbol whose kind is to be returned.
: 4069 4177 1
: 4070 4178 1     KIND - The address of a longword location where the symbol's kind
: 4071 4179 1           code should be returned.
: 4072 4180 1
: 4073 4181 1 OUTPUTS
: 4074 4182 1     KIND - The 'kind' of the SYMID symbol is returned to KIND. This is
: 4075 4183 1           a small integer with the following possible values:
: 4076 4184 1
: 4077 4185 1           RST$K_MODULE    -- SYMID is a Module
: 4078 4186 1           RST$K_ROUTINE   -- SYMID is a Routine
: 4079 4187 1           RST$K_BLOCK    -- SYMID is a Block
: 4080 4188 1           RST$K_ENTRY    -- SYMID is an Entry Point
: 4081 4189 1           RST$K_LABEL    -- SYMID is a Label
: 4082 4190 1           RST$K_LINE    -- SYMID is a Line
: 4083 4191 1           RST$K_DATA    -- SYMID is a Data Item
: 4084 4192 1           RST$K_TYPE    -- SYMID is a Data Type
: 4085 4193 1
: 4086 4194 1     No value is returned by DBG$STA_SYMKIND.
: 4087 4195 1
: 4088 4196 1 BEGIN
: 4089 4197 2
: 4090 4198 2 MAP
: 4091 4199 2     SYMID: REF RST$ENTRY,      | Pointer to the RST entry whose 'kind'
: 4092 4200 2                     | is to be returned.
: 4093 4201 2     KIND: REF VECTOR[1];    | Pointer to the location where the
: 4094 4202 2                     | kind is to be returned.
: 4095 4203 2
: 4096 4204 2
: 4097 4205 2
: 4098 4206 2
: 4099 4207 2     ! Make sure SYMID points to a valid RST entry (or at least seems to). Then
: 4100 4208 2     ! copy the entry's kind to KIND and return.
: 4101 4209 2
: 4102 4210 2     IF .SYMID[RST$B_KIND] LEQ RST$K_KIND_MINIMUM OR
: 4103 4211 2     .SYMID[RST$B_KIND] GTR RST$K_KIND_MAXIMUM
: 4104 4212 2     THEN
: 4105 4213 2         $DBG_ERROR('RSTACCESS\SYMKIND');
: 4106 4214 2
: 4107 4215 2     KIND[0] = .SYMID[RST$B_KIND];
: 4108 4216 2     RETURN;
: 4109 4217 2
: 4110 4218 1 END;
```

RSTACCESS
V04-000

I 11
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 130
(26)

4B 4D 59 53 5C 53 53 45 43 43 41 54 53 52 11 001D9 P.ABA: .ASCII <17>\RSTACCESS\<92>\SYMKIND\
44 4E 49 001E8

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

0004 00000
50 04 AC D0 00002
52 14 A0 9A 00006
05 15 0000A
0D 52 91 0000C
15 1B 0000F
00000000' EF 9F 00011 1\$:
01 DD 00017
00028362 8F DD 00019
00000000G 00 03 FB 0001F
08 BC 52 D0 00026 2\$:
04 0002A

.ENTRY DBG\$STA SYMKIND, Save R2
MOVL SYMID, R0
MOVZBL 20(R0), R2
BLEQ 1\$
CMPB R2, #13
BLEQU 2\$
PUSHAB P.ABA
PUSHL #1
PUSHL #164706
CALLS #3, LIB\$SIGNAL
MOVL R2, @KIND
RET

: 4165
: 4210
:
:
: 4211
:
: 4213
:
:
: 4215
: 4218

; Routine Size: 43 bytes, Routine Base: DBG\$CODE + 1C06


```
4112 4219 1 GLOBAL ROUTINE DBG$STA_SYMNAME(SYMID, NAMEPTR): NOVALUE =
4113 4220 1
4114 4221 1 FUNCTION:
4115 4222 1 This routine accepts a symbol identifier and returns the corresponding
4116 4223 1 symbol's name without any qualification. The symbol identifier is the
4117 4224 1 unique identifier produced by DBG$STA_GETSYMBOL or DBG$STA_GETSYMOFF.
4118 4225 1 The returned symbol name is represented as a counted ASCII string.
4119 4226 1
4120 4227 1 Since this routine does not produce a completely qualified, unambiguous
4121 4228 1 name, it is primarily used to get the names of data record components.
4122 4229 1 Such component names are needed by language-specific routines when
4123 4230 1 printing the values of data records.
4124 4231 1
4125 4232 1 INPUTS:
4126 4233 1 SYMID - A longword symbol identifier previously produced by routine
4127 4234 1 DBG$STA_GETSYMBOL or DBG$STA_GETSYMOFF. SYMID uniquely ident-
4128 4235 1 ifies the symbol whose name is to be returned.
4129 4236 1
4130 4237 1 NAMEPTR - The address of a longword location where a pointer to the
4131 4238 1 symbol's name should be returned.
4132 4239 1
4133 4240 1 OUTPUTS:
4134 4241 1 NAMEPTR - A pointer to the counted ASCII string giving the symbol's
4135 4242 1 bottom level, unqualified name is returned to NAMEPTR.
4136 4243 1
4137 4244 1 No value is returned by DBG$STA_SYMNAME.
4138 4245 1
4139 4246 1
4140 4247 2 BEGIN
4141 4248 2
4142 4249 2 MAP
4143 4250 2 SYMID: REF RST$ENTRY, ! Pointer to the RST entry whose name
4144 4251 2 string is to be returned.
4145 4252 2 NAMEPTR: REF VECTOR[1]; ! Pointer to the location where the
4146 4253 2 string address is to be returned.
4147 4254 2
4148 4255 2
4149 4256 2
4150 4257 2 ! Make sure SYMID seems to point to a valid RST entry. Copy the address
4151 4258 2 ! of the name string to NAMEPTR by calling GET_DST_NAME. Then return.
4152 4259 2
4153 4260 2 IF .SYMID[RST$B_KIND] LSS RST$K_KIND_MINIMUM OR
4154 4261 2 .SYMID[RST$B_KIND] GTR RST$K_KIND_MAXIMUM
4155 4262 2 THEN
4156 4263 2 $DBG_ERROR('RSTACCESS\SYMNAME');
4157 4264 2
4158 4265 2 NAMEPTR[0] = DBG$GET_DST_NAME(.SYMID[RST$L_DSTPTR]);
4159 4266 2 RETURN;
4160 4267 2
4161 4268 1 END;
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

4E 4D 59 53 5C 53 53 45 43 43 41 54 53 52 11 001EB P.ABB: .ASCII <17>\RSTACCESS\<92>\SYMNAME\
45 4D 41 001FA

				0004 00000	.PSECT	DBG\$CODE,NOWRT, SHR, PIC,0	
	52	04	AC	D0 00002	.ENTRY	DBG\$STA SYMNAME, Save R2	: 4219
	0D	14	A2	91 00006	MOVL	SYMID, R2	: 4261
			15	1B 0000A	CMPB	20(R2), #13	:
		00000000'	EF	9F 0000C	BLEQU	1\$:
			01	DD 00012	PUSHAB	P.ABB	: 4263
		00028362	8F	DD 00014	PUSHL	#1	:
00000000G	00		03	FB 0001A	PUSHL	#164706	:
		0C	A2	DD 00021	CALLS	#3, LIB\$SIGNAL	:
00000000G	00		01	FB 00024	PUSHL	12(R2)	: 4265
08	BC		50	D0 0002B	CALLS	#1, DBG\$GET DST_NAME	:
			04	0002F	MOVL	R0, @NAMEPTR	:
					RET		: 4268

; Routine Size: 48 bytes. Routine Base: DBG\$CODE + 1C31

```
4163 4269 1 GLOBAL ROUTINE DBG$STA_SYMPARENT(SYMID) =
4164 4270 1
4165 4271 1 FUNCTION
4166 4272 1     This routine returns the parent data item of a record (structure) compo-
4167 4273 1     nent. Thus, if there is a data item A.B(2).C, then the parent of C is
4168 4274 1     B and the parent of B is A. A does not have any parent. This routine
4169 4275 1     should only be called when the data component has been looked up direct-
4170 4276 1     ly in the RST via DBG$STA_GETSYMBOL, as would be done in languages like
4171 4277 1     PL/I or Cobol where record qualification need not be explicitly stated.
4172 4278 1
4173 4279 1 INPUTS
4174 4280 1     SYMID - The SYMID returned by DBG$STA_GETSYMBOL for the data item
4175 4281 1     whose parent data item is to be found. This symbol must
4176 4282 1     be of kind RST$K_DATA.
4177 4283 1
4178 4284 1 OUTPUTS
4179 4285 1     The SYMID of the input symbol's parent symbol is returned as the routine
4180 4286 1     value. If the input symbol does not have a parent, i.e. if
4181 4287 1     the input symbol is not a record component but a separate data
4182 4288 1     item in its own right, zero is returned as the routine value.
4183 4289 1
4184 4290 1
4185 4291 2 BEGIN
4186 4292 2
4187 4293 2 MAP
4188 4294 2     SYMID: REF RST$ENTRY;           ! Pointer to input symbol's RST entry
4189 4295 2
4190 4296 2 LOCAL
4191 4297 2     RSTPTR: REF RST$ENTRY;         ! Pointer to the first up-scope symbol
4192 4298 2     ! --this may be the parent symbol
4193 4299 2
4194 4300 2
4195 4301 2
4196 4302 2 ! Make sure the input parameter is the SYMID of a Data Item RST Entry.
4197 4303 2
4198 4304 2 IF .SYMID[RST$B_KIND] NEQ RST$K_DATA
4199 4305 2 THEN
4200 4306 2     $DBG_ERROR('RSTACCESS\SYMPARENT');
4201 4307 2
4202 4308 2
4203 4309 2 ! Get the first RST entry up-scope from the input symbol. If this is a Data
4204 4310 2 ! Item RST Entry, return its SYMID as the routine value. Otherwise, return
4205 4311 2 ! a zero as the routine value.
4206 4312 2
4207 4313 2 RSTPTR = .SYMID[RST$L_UPSCOPEPTR];
4208 4314 2 IF .RSTPTR[RST$B_KIND] EQL RST$K_DATA THEN RETURN .RSTPTR;
4209 4315 2 RETURN 0;
4210 4316 2
4211 4317 1 END;
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

```
50 4D 59 53 5C 53 53 45 43 43 41 54 53 52 13 001FD P.ABC: .ASCII <19>\RSTACCESS\<92>\SYMPARENT\
54 4E 45 52 41 0020C
```

:

				0004 00000	.PSECT	DBG\$CODE,NOWRT, SHR, PIC,0	
	52	04	AC	D0 00002	.ENTRY	DBG\$STA SYMPARENT, Save R2	: 4269
	06	14	A2	91 00006	MOVL	SYMID, R2	: 4304
			15	13 0000A	CMPB	20(R2), #6	:
		00000000'	EF	9F 0000C	BEQL	1\$:
			01	DD 00012	PUSHAB	P.ABC	: 4306
		00028362	8F	DD 0C014	PUSHL	#1	:
00000000G	00		03	FB 0001A	PUSHL	#164706	:
	50	10	A2	D0 00021 1\$:	CALLS	#3, LIB\$SIGNAL	:
	06	14	A0	91 00025	MOVL	16(R2), RSTPTR	: 4313
			02	13 00029	CMPB	20(RSTPTR), #6	: 4314
			50	D4 0002B	BEQL	2\$:
			04	0002D 2\$:	CLRL	R0	: 4315
					RET		: 4317

; Routine Size: 46 bytes, Routine Base: DBG\$CODE + 1C61

```
4213 4318 1 GLOBAL ROUTINE DBG$STA_SYMPATHNAME(SYMID, PATHNAME): NOVALUE =
4214 4319 1
4215 4320 1 FUNCTION
4216 4321 1 This routine accepts a symbol identifier and returns the corresponding
4217 4322 1 symbol's fully qualified pathname. The symbol identifier is the unique
4218 4323 1 identifier produced by the DBG$STA_GETSYMBOL or DBG$STA_GETSYMOFF rou-
4219 4324 1 tine. The returned pathname is represented in internal-format by a
4220 4325 1 pathname descriptor which includes the symbol name with all possible
4221 4326 1 pathname qualification and all possible data record qualification. This
4222 4327 1 does not include array subscripts, however.
4223 4328 1
4224 4329 1 This routine is called when a symbol's name is to be printed in a com-
4225 4330 1 pletely unambiguous form. The returned pathname is not in a directly
4226 4331 1 printable form, but can relatively easily be converted to a character
4227 4332 1 string by language-specific routines.
4228 4333 1
4229 4334 1 INPUTS
4230 4335 1 SYMID - A longword symbol identifier previously produced by routine
4231 4336 1 DBG$STA_GETSYMBOL or DBG$STA_GETSYMOFF. SYMID uniquely identi-
4232 4337 1 fies the symbol whose complete pathname is to be returned.
4233 4338 1
4234 4339 1 PATHNAME - The address of a longword location where a pointer to the
4235 4340 1 symbol's pathname descriptor should be returned.
4236 4341 1
4237 4342 1 OUTPUTS
4238 4343 1 PATHNAME - A full pathname descriptor for the SYMID symbol is generated
4239 4344 1 and a pointer to that descriptor is returned to PATHNAME. The
4240 4345 1 descriptor will disappear after the processing of the current
4241 4346 1 DEBUG command.
4242 4347 1
4243 4348 1 No value is returned by DBG$STA_SYMPATHNAME.
4244 4349 1
4245 4350 1
4246 4351 2 BEGIN
4247 4352 2
4248 4353 2 MAP
4249 4354 2 SYMID: REF RST$ENTRY, : Pointer to input RST entry
4250 4355 2 PATHNAME: REF VECTOR[1]; : Pointer to returned pathname descr.
4251 4356 2
4252 4357 2 LOCAL
4253 4358 2 COMPCNT, : Number of data components in pathname
4254 4359 2 DATACNT, : Number of Data RST Entries in chain
4255 4360 2 INVOC_LOC, : Location in NAMELIST where invocation
4256 4361 2 : number belongs (inner-most rout-
4257 4362 2 : ine in SYMID's environment)
4258 4363 2 INVOCNUM, : The invocation number itself
4259 4364 2 INVPTR: REF RST$ENTRY, : Pointer to Invocation Number RST Entry
4260 4365 2 J : Pathname vector index
4261 4366 2 LINE_END, : Line end address (not actually used)
4262 4367 2 LINE_LWRDS, : Number of longwords needed for line
4263 4368 2 : number counted ASCII string
4264 4369 2 LINE_NUM, : The line number used to identify an
4265 4370 2 : anonymous lexical entity
4266 4371 2 LINE_NUM_FOUND, : Set to TRUE if a line number RST entry
4267 4372 2 : is in the symbol's up-scope chain
4268 4373 2 LINE_NUM_LOC, : Location in NAMELIST before which the
4269 4374 2 : line number should be inserted
```

```
: 4270      4375 2      LINE_NUM_PTR: REF VECTOR[.BYTE], ! Pointer to line number counted ASCII
: 4271      4376 2      LINE_START, ! Line start address (not actually used)
: 4272      4377 2      LINE_STRING: VECTOR[40,BYTE], ! Vector used to build ASCII line number
: 4273      4378 2      LSI, ! Index of next location in LINE_STRING
: 4274      4379 2      MODPTR, ! Module RST pointer (not actually used)
: 4275      4380 2      NAMECNT, ! The number of pathname components
: 4276      4381 2      NAMELIST: VECTOR[DBG$K_MAX_PATHNAME], ! Vector of pointers to names
: 4277      4382 2      NAMEPTR: REF VECTOR[.BYTE], ! Pointer to current pathname component
: 4278      4383 2      ! (as a counted ASCII string)
: 4279      4384 2      NO_NULL_NAME, ! Set to TRUE if no null lexical entity
: 4280      4385 2      ! name is in up-scope chain
: 4281      4386 2      NO_ROUTINE, ! Set to TRUE if inner-most routine has
: 4282      4387 2      ! not yet been found up-scope
: 4283      4388 2      PATHDESCR: REF PTH$PATHNAME, ! Pointer to Pathname Descriptor
: 4284      4389 2      PATHVEC: REF VECTOR[.LONG], ! Pointer to pathname vector in descr.
: 4285      4390 2      RSTPTR: REF RST$ENTRY, ! Pointer to current RST entry
: 4286      4391 2      STATUS, ! Status code returned by called routine
: 4287      4392 2      STMT_NUM; ! Statement number within line number
: 4288      4393 2
: 4289      4394 2
: 4290      4395 2
: 4291      4396 2      ! Initialize some pointers and counters for the up-scope chain loop.
: 4292      4397 2
: 4293      4398 2      RSTPTR = .SYMID;
: 4294      4399 2      NAMECNT = 0;
: 4295      4400 2      DATACNT = 0;
: 4296      4401 2      LINE_NUM_FOUND = FALSE;
: 4297      4402 2      LINE_NUM_LOC = 1000000;
: 4298      4403 2      NO_NULL_NAME = TRUE;
: 4299      4404 2      NO_ROUTINE = TRUE;
: 4300      4405 2
: 4301      4406 2
: 4302      4407 2      ! Go up the input symbol's up-scope chain to determine how many pathname
: 4303      4408 2      ! components the symbol has. We also determine how much data qualification
: 4304      4409 2      ! there is and whether a line number needs to be supplied in the pathname.
: 4305      4410 2
: 4306      4411 2      WHILE TRUE DO
: 4307      4412 3      BEGIN
: 4308      4413 3
: 4309      4414 3
: 4310      4415 3      ! Get the name of the pathname component. Unless the name is null,
: 4311      4416 3      ! save a pointer to the name string in the NAMELIST vector.
: 4312      4417 3
: 4313      4418 3      NAMEPTR = DBG$GET_DST_NAME(.RSTPTR[RST$DSTPTR]);
: 4314      4419 3      IF .NAMEPTR[0] NEQ 0
: 4315      4420 3      THEN
: 4316      4421 4      BEGIN
: 4317      4422 4      IF .NAMECNT GEQ DBG$K_MAX_PATHNAME THEN EXITLOOP;
: 4318      4423 4      NAMELIST[.NAMECNT] = .NAMEPTR;
: 4319      4424 4      NAMECNT = .NAMECNT + 1;
: 4320      4425 3      END;
: 4321      4426 3
: 4322      4427 3
: 4323      4428 3      ! If this is a global symbol, exit the up-scope loop right away.
: 4324      4429 3
: 4325      4430 3      IF .RSTPTR[RST$V_GLOBAL] THEN EXITLOOP;
: 4326      4431 3
```

4327 4432 3
4328 4433 3
4329 4434 3
4330 4435 3
4331 4436 3
4332 4437 3
4333 4438 3
4334 4439 3
4335 4440 3
4336 4441 3
4337 4442 3
4338 4443 4
4339 4444 4
4340 4445 5
4341 4446 4
4342 4447 5
4343 4448 5
4344 4449 5
4345 4450 4
4346 4451 4
4347 4452 4
4348 4453 4
4349 4454 5
4350 4455 5
4351 4456 5
4352 4457 5
4353 4458 5
4354 4459 5
4355 4460 5
4356 4461 4
4357 4462 4
4358 4463 3
4359 4464 3
4360 4465 3
4361 4466 3
4362 4467 3
4363 4468 3
4364 4469 3
4365 4470 3
4366 4471 3
4367 4472 3
4368 4473 3
4369 4474 3
4370 4475 3
4371 4476 3
4372 4477 3
4373 4478 3
4374 4479 3
4375 4480 3
4376 4481 3
4377 4482 3
4378 4483 3
4379 4484 3
4380 4485 3
4381 4486 3
4382 4487 3
4383 4488 2

```
! Determine what kind of RST entry this is and act accordingly.
CASE .RSTPTR[RST$B_KIND] FROM RST$K_KIND_MINIMUM TO RST$K_KIND_MAXIMUM OF
SET
  [RST$K_MODULE]:
    EXITLOOP;
  [RST$K_ROUTINE,
   RST$K_BLOCK]:
    BEGIN
      IF .NO_ROUTINE AND (.NAMEPTR[0] NEQ 0) AND
        (.RSTPTR[RST$B_KIND] EQL RST$K_ROUTINE)
      THEN
        BEGIN
          NO_ROUTINE = FALSE;
          INVOC_LOC = .NAMECNT - 1;
        END;
      IF (.NAMEPTR[0] EQL 0) AND .NO_NULL_NAME
      THEN
        BEGIN
          LINE_NUM_LOC = .NAMECNT;
          MODPTR = .RSTPTR;
          IF DBG$PC_TO_LINE_LOOKUP(.RSTPTR[RST$L_STARTADDR],
                                   LINE_NUM, STMT_NUM,
                                   LINE_START, LINE_END, MODPTR)
          THEN NO_NULL_NAME = FALSE;
        END;
      END;
  [RST$K_ENTRY,
   RST$K_OVERLOAD,
   RST$K_LABEL]:
    0;
  [RST$K_LINE]:
    LINE_NUM_FOUND = TRUE;
  [RST$K_DATA,
   RST$K_TYPE,
   RST$K_TYPCOMP]:
    IF .NAMEPTR[0] NEQ 0 THEN DATAcnt = .DATAcnt + 1;
  [INRANGE]:
    $DBG_ERROR('RSTACCESS\SYMPATHNAME');
TES;

! Link to the next RST entry up-scope from this one. Then loop.
RSTPTR = .RSTPTR[RST$L_UPSCOPEPTR];
END;
```

```

: 4384      4489 2
: 4385      4490 2
: 4386      4491 2
: 4387      4492 2
: 4388      4493 2
: 4389      4494 2
: 4390      4495 2
: 4391      4496 2
: 4392      4497 2
: 4393      4498 2
: 4394      4499 2
: 4395      4500 2
: 4396      4501 2
: 4397      4502 2
: 4398      4503 2
: 4399      4504 2
: 4400      4505 2
: 4401      4506 2
: 4402      4507 2
: 4403      4508 2
: 4404      4509 2
: 4405      4510 2
: 4406      4511 2
: 4407      4512 3
: 4408      4513 3
: 4409      4514 3
: 4410      4515 3
: 4411      4516 3
: 4412      4517 3
: 4413      4518 3
: 4414      4519 3
: 4415      4520 4
: 4416      4521 4
: 4417      4522 5
: 4418      4523 5
: 4419      4524 5
: 4420      4525 5
: 4421      4526 4
: 4422      4527 4
: 4423      4528 4
: 4424      4529 4
: 4425      4530 3
: 4426      4531 3
: 4427      4532 3
: 4428      4533 3
: 4429      4534 3
: 4430      4535 3
: 4431      4536 4
: 4432      4537 4
: 4433      4538 4
: 4434      4539 4
: 4435      4540 3
: 4436      4541 3
: 4437      4542 3
: 4438      4543 3
: 4439      4544 3
: 4440      4545 3

! Determine how many levels of data qualification (e.g., 2 for M\A.B.C)
! there is in the pathname.
IF .DATACNT EQL 0
THEN
  COMPCNT = 0
ELSE
  COMPCNT = .DATACNT - 1;

! If there already is a line number in the pathname, do not insert an extra
! line number due to a null lexical entity name.
IF .NO_NULL_NAME OR .LINE_NUM_FOUND THEN LINE_NUM_LOC = 1000000;

! If we do have to supply a line number in the pathname to identify an
! anonymous lexical entity, generate the line number counted ASCII string.
LINE_LWRDS = 0;
IF .LINE_NUM_LOC NEQ 1000000
THEN
  BEGIN
    LSI = 0;

    ! If there is a statement number, convert that to ASCII decimal.
    IF .STMT_NUM NEQ 0
    THEN
      BEGIN
        WHILE .STMT_NUM NEQ 0 DO
          BEGIN
            LINE_STRING[.LSI] = (.STMT_NUM MOD 10) + '0';
            LSI = .LSI + 1;
            STMT_NUM = .STMT_NUM/10;
          END;

          LINE_STRING[.LSI] = '.';
          LSI = .LSI + 1;
        END;

        ! Convert the main statement number to ASCII decimal.
        WHILE .LINE_NUM NEQ 0 DO
          BEGIN
            LINE_STRING[.LSI] = (.LINE_NUM MOD 10) + '0';
            LSI = .LSI + 1;
            LINE_NUM = .LINE_NUM/10;
          END;

          ! Compute the number of longwords we will need for the line number.
          LINE_LWRDS = (.LSI + 13)/4;

```



```

4441 4546 2      END;
4442 4547 2
4443 4548 2
4444 4549 2      ! Determine what the invocation number is. If it turns out to be zero,
4445 4550 2      ! we do not explicitly put it in the Pathname Descriptor.
4446 4551 2
4447 4552 2      INVOCNUM = 0;
4448 4553 2      IF .SYMID[RST$V_INVOCNUM]
4449 4554 2      THEN
4450 4555 2          BEGIN
4451 4556 2              INVPTR = .SYMID[RST$L_SYMCHNPTR];
4452 4557 2              INVOCNUM = .INVPTR[RST$L_INVOCNUM];
4453 4558 2          END;
4454 4559 2
4455 4560 2      IF .INVOCNUM EQL 0 THEN INVOC_LOC = 1000000;
4456 4561 2
4457 4562 2
4458 4563 2      ! Allocate space for a Pathname Descriptor for the symbol.
4459 4564 2
4460 4565 2      PATHDESCR = DBG$GET_TEMPMEM(DBG$K_PATHDESCSIZE + .NAMECNT + .LINE_LWRDS);
4461 4566 2      PATHVEC = PATHDESCR[PTH$A_PATHVECTOR];
4462 4567 2
4463 4568 2
4464 4569 2      ! Fill in the Pathname Descriptor's header.
4465 4570 2
4466 4571 2      PATHDESCR[PTH$B_TOTCNT] = .NAMECNT;
4467 4572 2      PATHDESCR[PTH$B_PATHCNT] = .NAMECNT - .COMPCNT;
4468 4573 2      PATHDESCR[PTH$B_LOCINVOC] = 0;
4469 4574 2      PATHDESCR[PTH$L_INVOCNUM] = 0;
4470 4575 2
4471 4576 2
4472 4577 2      ! Fill in the pointers to the pathname component names.
4473 4578 2
4474 4579 2      J = 0;
4475 4580 2      DECR I FROM .NAMECNT - 1 TO 0 DO
4476 4581 2          BEGIN
4477 4582 2              PATHVEC[.J] = .NAMELIST[.I];
4478 4583 2              J = .J + 1;
4479 4584 2
4480 4585 2
4481 4586 2      ! If this is where the invocation number goes, mark that in the header.
4482 4587 2
4483 4588 2      IF .I EQL .INVOC_LOC
4484 4589 2      THEN
4485 4590 2          BEGIN
4486 4591 2              PATHDESCR[PTH$B_LOCINVOC] = .J;
4487 4592 2              PATHDESCR[PTH$L_INVOCNUM] = .INVOCNUM;
4488 4593 2          END;
4489 4594 2
4490 4595 2
4491 4596 2      ! If this is where the extra line number goes, fill that in.
4492 4597 2
4493 4598 2      IF .J EQL .LINE_NUM_LOC
4494 4599 2      THEN
4495 4600 2          BEGIN
4496 4601 2              LINE_NUM_PTR = PATHVEC[.NAMECNT + 1];
4497 4602 2              LINE_NUM_PTR[0] = .LSI + 6;
```

```
4498 4603 4 LINE_NUM_PTR[1] = 'X';
4499 4604 4 LINE_NUM_PTR[2] = 'L';
4500 4605 4 LINE_NUM_PTR[3] = 'I';
4501 4606 4 LINE_NUM_PTR[4] = 'N';
4502 4607 4 LINE_NUM_PTR[5] = 'E';
4503 4608 4 LINE_NUM_PTR[6] = ' ';
4504 4609 4 INCR K FROM 1 TO .LSI DO
4505 4610 4     LINE_NUM_PTR[.K + 6] = .LINE_STRING[.LSI - .K];
4506 4611 4
4507 4612 4 PATHVEC[.J] = .LINE_NUM_PTR;
4508 4613 4 J = .J + 1;
4509 4614 3 END;
4510 4615 3
4511 4616 2 END;
4512 4617 2
4513 4618 2
4514 4619 2 ! Finally return the address of the Pathname Descriptor to PATHNAME. Then
4515 4620 2 ! return.
4516 4621 2
4517 4622 2 PATHNAME[0] = .PATHDESCR;
4518 4623 2 RETURN;
4519 4624 2
4520 4625 1 END;
```

```
50 4D 59 53 5C 53 53 45 43 43 41 54 53 52 15 00211 P.ABD: .PSECT DBG$PLIT,NOWRT, SHR, PIC,0
45 4D 41 4E 48 54 41 00220 .ASCII <21>\RSTACCESS\<92>\SYMPATHNAME\
```

```
                                .PSECT DBG$CODE,NOWRT, SHR, PIC,0
                                .ENTRY DBG$STA SYMPATHNAME, Save R2,R3,R4,R5,R6,-
                                R7,R8,R9,R10,R11
                                MOVAB -260(SP), SP
                                MOVL SYMID, R5
                                MOVL R5, RSTPTR
                                CLRL NAMECNT
                                CLRL LINE_NUM_FOUND
                                MOVL #1000000, LINE_NUM_LOC
                                MOVQ #1, NO_NULL_NAME
                                MOVL #1, NO_ROUTINE
                                PUSHL 12(RSTPTR)
                                CALLS #1, DBG$GET_DST_NAME
                                MOVL R0, NAMEPTR
                                CLRL R0
                                TSTB (NAMEPTR)
                                BEQL 2$
                                INCL R0
                                CMPL NAMECNT, #50
                                BGEQ 4$
                                MOVL NAMEPTR, NAMELIST[NAMECNT]
                                INCL NAMECNT
                                BLBS 21(RSTPTR), 4$

                                OFFC 00000
                                5E FEFC CE 9E 00002
                                55 04 AC D0 00007
                                52 55 D0 0000B
                                53 53 D4 0000E
                                5A D4 00010
                                5B 000F4240 8F D0 00012
                                56 01 7D 00019
                                58 01 D0 0001C
                                00000000G 00 0C A2 DD 0001F 1$:
                                59 01 FB 00022
                                50 D0 00029
                                50 D4 0002C
                                69 95 0002E
                                0E 13 00030
                                50 D6 00032
                                32 53 D1 00034
                                2C 18 00037
                                14 AE43 59 D0 00039
                                53 D6 0003E
                                21 15 A2 E8 00040 2$:
```

```
                                .PSECT DBG$CODE,NOWRT, SHR, PIC,0
                                .ENTRY DBG$STA SYMPATHNAME, Save R2,R3,R4,R5,R6,-
                                R7,R8,R9,R10,R11
                                MOVAB -260(SP), SP
                                MOVL SYMID, R5
                                MOVL R5, RSTPTR
                                CLRL NAMECNT
                                CLRL LINE_NUM_FOUND
                                MOVL #1000000, LINE_NUM_LOC
                                MOVQ #1, NO_NULL_NAME
                                MOVL #1, NO_ROUTINE
                                PUSHL 12(RSTPTR)
                                CALLS #1, DBG$GET_DST_NAME
                                MOVL R0, NAMEPTR
                                CLRL R0
                                TSTB (NAMEPTR)
                                BEQL 2$
                                INCL R0
                                CMPL NAMECNT, #50
                                BGEQ 4$
                                MOVL NAMEPTR, NAMELIST[NAMECNT]
                                INCL NAMECNT
                                BLBS 21(RSTPTR), 4$

                                4318
                                4398
                                4399
                                4401
                                4402
                                4403
                                4404
                                4418
                                4419
                                4422
                                4423
                                4424
                                4430
```

001E	001E	0084	14	A2	8F	00044	CASEB	20(RSTPTR), #0, #13	4435
0061	0061	005C		0068		00049	.WORD	98-38,-	
0068	0061	0068		0C7D		00051		118-38,-	
		007D		007D		00059		58-38,-	
				0068		00061		58-38,-	
								108-38,-	
								78-38,-	
								88-38,-	
								88-38,-	
								108-38,-	
								98-38,-	
								88-38,-	
								98-38,-	
								98-38,-	
								108-38	
				66	11	00065	BRB	118	4439
		0F		58	E9	00067	BLBC	NO_ROUTINE, 68	4444
		0C		50	E9	0006A	BLBC	R0, 68	
		02	14	A2	91	0006D	CMQB	20(RSTPTR), #2	4445
				06	12	00071	BNEQ	68	
				58	D4	00073	CLRL	NO_ROUTINE	4448
		54	FF	A3	9E	00075	MOVAB	-1(R3), INVOC_LOC	4449
				69	95	00079	TSTB	(NAMEPTR)	4452
				49	12	0007B	BNEQ	108	
		46		56	E9	0007D	BLBC	NO_NULL_NAME, 108	
		58		53	D0	00080	MOVL	NAMECNT, LINE_NUM_LOC	4455
		6E		52	D0	00083	MOVL	RSTPTR, MODPTR	4456
				5E	DD	00086	PUSHL	SP	4457
			08	AE	9F	00C88	PUSHAB	LINE_END	
			10	AE	9F	0008B	PUSHAB	LINE_START	
			18	AE	9F	0008E	PUSHAB	STMT_NUM	
			20	AE	9F	00091	PUSHAB	LINE_NUM	
			18	A2	DD	00094	PUSHL	24(RSTPTR)	
00000000G	00			06	FB	00097	CALLS	#6, DBG\$PC_TO_LINE_LOOKUP	
	25			50	E9	0009E	BLBC	R0, 108	
				56	D4	000A1	CLRL	NO_NULL_NAME	4460
				21	11	000A3	BRB	108	4465
	5A			01	D0	000A5	MOVL	#1, LINE_NUM_FOUND	4471
				1C	11	000A8	BRB	108	
	19			50	E9	000AA	BLBC	R0, 108	4476
				57	D6	000AD	INCL	DATACNT	
				15	11	000AF	BRB	108	
		00000000'		EF	9F	000B1	PUSHAB	P.ABD	4479
				01	DD	000B7	PUSHL	#1	
		00028362		8F	DD	000B9	PUSHL	#164706	
00000000G	00			03	FB	000BF	CALLS	#3, LIB\$SIGNAL	
	52		10	A2	D0	000C6	MOVL	16(RSTPTR), RSTPTR	4486
				FF	52	31	BRW	18	4411
				57	D5	000CD	TSTL	DATACNT	4493
				04	12	000CF	BNEQ	128	
				58	D4	000D1	CLRL	COMPCNT	4495
				04	11	000D3	BRB	138	
	58		FF	A7	9E	000D5	MOVAB	-1(R7), COMPCNT	4497
	03			56	E8	000D9	BLBS	NO_NULL_NAME, 148	4503
	07			5A	E9	000DC	BLBC	LINE_NUM_FOUND, 158	
	5B	000F4240		8F	D0	000DF	MOVL	#1000000, LINE_NUM_LOC	
				51	D4	000E6	CLRL	LINE_LWRDS	4509

		000F4240	8F	5B	D1	000E8	CMPL	LINE_NUM_LOC, #1000000	4510		
				4F	13	0C0EF	BEQL	20\$			
				52	D4	000F1	CLRL	LSI	4513		
			0C	AE	D5	000F3	TSTL	STMT_NUM	4518		
				22	13	000F6	BEQL	18\$			
				19	13	000F8	BEQL	17\$	4521		
7E	00	0C	AE	01	7A	000FA	EMUL	#1, STMT_NUM, #0, -(SP)	4523		
50	50		8E	0A	7B	00100	EDIV	#10, (SPT), R0, R0			
	D8	AD42	50	30	81	00105	ADDB3	#48, R0, LINE_STRING[LSI]			
				52	D6	0010B	INCL	LSI	4524		
			0C	0A	C6	0010D	DIVL2	#10, STMT_NUM	4525		
				E5	11	00111	BRB	16\$	4521		
			D8	2E	90	00113	MOVB	#46, LINE_STRING[LSI]	4528		
				52	D6	00118	INCL	LSI	4529		
				10	AE	D5	0011A	TSTL	LINE_NUM	4535	
				19	13	0011D	BEQL	19\$			
7E	00	10	AE	01	7A	0011F	EMUL	#1, LINE_NUM, #0, -(SP)	4537		
50	50		8E	0A	7B	00125	EDIV	#10, (SPT), R0, R0			
	D8	AD42	50	30	81	0012A	ADDB3	#48, R0, LINE_STRING[LSI]			
				52	D6	00130	INCL	LSI	4538		
			10	0A	C6	00132	DIVL2	#10, LINE_NUM	4539		
				E2	11	00136	BRB	18\$	4535		
			56	0D	A2	9E	00138	MOVAB	13(R2), R6	4545	
	51		56		04	C7	0013C	DIVL3	#4, R6, LINE_LWRDS		
				5A	D4	00140	CLRL	INVOCNUM	4552		
	08	15	A5	02	E1	00142	BBC	#2, 21(R5), 21\$	4553		
			50	08	A5	D0	00147	MOVL	8(R5), INVPTR	4556	
			5A	18	A0	D0	0014B	MOVL	24(INVPTR), INVOCNUM	4557	
				07	12	0014F	BNEQ	22\$	4560		
		54	000F4240	8F	D0	00151	MOVL	#1000000, INVOC_LOC			
				02	A143	9F	00158	PUSHAB	2(LINE_LWRDS)[NAMECNT]	4565	
		00000000G	00	01	FB	0015C	CALLS	#1, DBG\$GET_TEMPMEM			
			59	08	A0	9E	00163	MOVAB	8(R0), PATHVEC	4566	
			60	53	90	00167	MOVB	NAMECNT, (PATHDESCR)	4571		
01	A0		53	58	83	0016A	SUSB3	COMPCNT, NAMECNT, 1(PATHDESCR)	4572		
				02	A0	94	0016F	CLRB	2(PATHDESCR)	4573	
				04	A0	D4	00172	CLRL	4(PATHDESCR)	4574	
				57	D4	00175	CLRL	J	4579		
			51	53	D0	00177	MOVL	NAMECNT, I	4588		
				4A	11	0017A	BRB	27\$			
		6947		14	AE41	D0	0017C	MOVL	NAMELIST[I], (PATHVEC)[J]	4582	
				57	D6	00182	INCL	J	4583		
			54	51	D1	00184	CMPL	I, INVOC_LOC	4588		
				08	12	00187	BNEQ	24\$			
			02	57	90	00189	MOVB	J, 2(PATHDESCR)	4591		
			04	5A	D0	0018D	MOVL	INVOCNUM, 4(PATHDESCR)	4592		
			5B	57	D1	00191	CMPL	J, LINE_NUM_LOC	4598		
				30	12	00194	BNEQ	27\$			
			55	04	A943	DE	00196	MOVAL	4(PATHVEC)[NAMECNT], LINE_NUM_PTR	4601	
65			52	06	81	0019B	ADDB3	#6, LSI, (LINE_NUM_PTR)	4602		
			01	A5	4E494C25	8F	D0	0019F	MOVL	#1313426469, 1(LINE_NUM_PTR)	4603
			05	A5	2045	8F	B0	001A7	MOVW	#8261, 5(LINE_NUM_PTR)	4607
				58	D4	001AD	CLRL	K	4609		
				0B	11	001AF	BRB	26\$			
56			52	58	C3	001B1	SUBL3	K, LSI, R6	4610		
			06	A845	D8	AD46	MOVB	LINE_STRING[R6], 6(K)[LINE_NUM_PTR]			
F1			58	52	F3	001BC	AOBLEQ	LSI, K, 25\$			

```

1 12
16-Sep-1984 02:48:17      YAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26      [DEBUG.SRC]RSTACCESS.B32;1

```

Page 143
(29)

RV

```

08 6947 55 D0 001C0      MOVL    LINE_NUM_PTR, (PATHVEC)[J]
      57 D6 001C4      INCL    J
      B3 51 F4 001C6 27$ SOBGEQ    I, 23$
      BC 50 D0 001C9      MOVL    PATHDESCR, @PATHNAME
      04 001CD      RET

```

: 4612
: 4613
: 4580
: 4622
: 4625

; Routine Size: 462 bytes, Routine Base: DBG\$CODE + 1C8F

```
4522 4626 1 GLOBAL ROUTINE DBG$STA_SYMVALUE(SYMID, VALPTR, VALKIND): NOVALUE =
4523 4627 1
4524 4628 1 FUNCTION
4525 4629 1     This routine accepts a symbol identifier and returns a pointer to the
4526 4630 1     corresponding symbol's value. The symbol identifier is the unique
4527 4631 1     identifier produced by routine DBG$STA_GETSYMBOL or DBG$STA_GETSYMOFF.
4528 4632 1
4529 4633 1     This routine requires a 'context' to have been established by a call on
4530 4634 1     routine DBG$STA_SETCONTEXT if there are any register references in the
4531 4635 1     DST Value Spec which defines the symbol's value. If such a reference
4532 4636 1     occurs and no context exists, an error is signalled.
4533 4637 1
4534 4638 1     The interpretation of the value stored at the returned address is up to
4535 4639 1     the language-specific routines in light of the symbol's data type. The
4536 4640 1     data type specification must therefore include all length information.
4537 4641 1
4538 4642 1 INPUTS
4539 4643 1     SYMID - A longword symbol identifier previously produced by routine
4540 4644 1           DBG$STA_GETSYMBOL or DBG$STA_GETSYMOFF. SYMID uniquely ident-
4541 4645 1           ifies the symbol whose 'value' is to be returned.
4542 4646 1
4543 4647 1     VALPTR - The address of a three-longword vector to receive the value
4544 4648 1           pointer and the corresponding stack frame pointer.
4545 4649 1
4546 4650 1     VALKIND - The address of a longword location to receive the value kind.
4547 4651 1
4548 4652 1 OUTPUTS
4549 4653 1     VALPTR - A pointer to the desired value is returned to VALPTR. The
4550 4654 1           byte address of the value is returned to VALPTR[0] and the
4551 4655 1           bit offset from that address is returned to VALPTR[1]. The
4552 4656 1           corresponding stack frame pointer is returned to VALPTR[2].
4553 4657 1           VALPTR[2] will contain zero if no frame pointer is applicable.
4554 4658 1
4555 4659 1     VALKIND - The kind of the value pointed to by VALPTR is returned to
4556 4660 1           VALKIND. These are the possible values:
4557 4661 1
4558 4662 1           DBG$K_VAL_NOVALUE - The symbol has no value.
4559 4663 1           DBG$K_VAL_LITERAL - VALPTR points to a literal value.
4560 4664 1           DBG$K_VAL_ADDR - VALPTR contains an address.
4561 4665 1           DBG$K_VAL_DESCR - VALPTR contains the address of a
4562 4666 1                           descriptor.
4563 4667 1
4564 4668 1     No value is returned by DBG$STA_SYMVALUE.
4565 4669 1
4566 4670 1
4567 4671 2 BEGIN
4568 4672 2
4569 4673 2 MAP
4570 4674 2     SYMID: REF RST$ENTRY,           ! Pointer to input symbol's RST entry
4571 4675 2     VALPTR: REF VECTOR[3],         ! Pointer to caller's value vector
4572 4676 2     VALKIND: REF VECTOR[1];        ! Pointer to value kind parameter
4573 4677 2
4574 4678 2 LOCAL
4575 4679 2     BLITRLR: REF DST$BLI_TRAILER1, ! Pointer to Bliss DST record trailer
4576 4680 2     BLIVALSPEC: BLOCK[8, BYTE]     ! Value Spec buffer for Bliss special
4577 4681 2     FIELD(DST$VS_HDR_FIELDS),      ! cases DST record
4578 4682 2     CH_TRLR_PTR: REF DST$CH_TRLR, ! Pointer to COBOL Hack DST trailer
```

```
: 4579      4683 2      DSTPTR: REF DST$RECORD,      ! Pointer to symbol's DST record
: 4580      4684 2      VALLOC: REF VECTOR[,LONG];      ! Value location from Stack Machine
: 4581      4685 2
: 4582      4686 2
: 4583      4687 2
: 4584      4688 2      ! Initially zero out the returned value pointer and frame pointer.
: 4585      4689 2
: 4586      4690 2      VALPTR[0] = 0;
: 4587      4691 2      VALPTR[1] = 0;
: 4588      4692 2      VALPTR[2] = 0;
: 4589      4693 2
: 4590      4694 2
: 4591      4695 2      ! Determine what kind of RST entry SYMID identifies and act accordingly.
: 4592      4696 2
: 4593      4697 2      CASE .SYMID[RST$B_KIND] FROM RST$K_KIND_MINIMUM TO RST$K_KIND_MAXIMUM OF
: 4594      4698 2          SET
: 4595      4699 2
: 4596      4700 2
: 4597      4701 2          ! For instruction addresses, return the start address in the RST entry.
: 4598      4702 2
: 4599      4703 2          [RST$K_ROUTINE, RST$K_BLOCK,
: 4600      4704 2          RST$K_ENTRY, RST$K_LABEL,
: 4601      4705 2          RST$K_LINE]:
: 4602      4706 3              BEGIN
: 4603      4707 3                  VALPTR[0] = .SYMID[RST$L_STARTADDR];
: 4604      4708 3                  VALKIND[0] = DBG$K_VAL_ADDR;
: 4605      4709 3                  RETURN;
: 4606      4710 2                  END;
: 4607      4711 2
: 4608      4712 2
: 4609      4713 2          ! For Types, return the No-Value code to VALKIND.
: 4610      4714 2
: 4611      4715 2          [RST$K_TYPE]:
: 4612      4716 3              BEGIN
: 4613      4717 3                  VALKIND[0] = DBG$K_VAL_NOVALUE;
: 4614      4718 3                  RETURN;
: 4615      4719 2                  END;
: 4616      4720 2
: 4617      4721 2
: 4618      4722 2          ! For most other kinds (including Module) signal an internal error.
: 4619      4723 2
: 4620      4724 2          [INRANGE, OVRANGE]:
: 4621      4725 2              $DBG_ERROR('RSTACCESS\SYMVALUE 10');
: 4622      4726 2
: 4623      4727 2
: 4624      4728 2          ! For Data and Type Components, do nothing here--we handle them below.
: 4625      4729 2
: 4626      4730 2          [RST$K_DATA, RST$K_TYPCOMP]:
: 4627      4731 2              0;
: 4628      4732 2
: 4629      4733 2          TES;
: 4630      4734 2
: 4631      4735 2
: 4632      4736 2          ! Obtain the DST record describing this object. If there are
: 4633      4737 2          ! continuation records then merge them into a single new DST record.
: 4634      4738 2
: 4635      4739 2      DSTPTR = .SYMID[RST$L_DSTPTR];
```

```
4636 4740 2
4637 4741 2
4638 4742 2
4639 4743 2
4640 4744 2
4641 4745 2
4642 4746 2
4643 4747 2
4644 4748 2
4645 4749 2
4646 4750 2
4647 4751 2
4648 4752 2
4649 4753 2
4650 4754 2
4651 4755 2
4652 4756 3
4653 4757 3
4654 4758 3
4655 4759 3
4656 4760 3
4657 4761 3
4658 4762 3
4659 4763 3
4660 4764 3
4661 4765 3
4662 4766 3
4663 4767 3
4664 4768 4
4665 4769 3
4666 4770 3
4667 4771 4
4668 4772 3
4669 4773 3
4670 4774 3
4671 4775 3
4672 4776 3
4673 4777 3
4674 4778 2
4675 4779 2
4676 4780 2
4677 4781 2
4678 4782 2
4679 4783 2
4680 4784 2
4681 4785 3
4682 4786 3
4683 4787 3
4684 4788 2
4685 4789 2
4686 4790 2
4687 4791 2
4688 4792 2
4689 4793 2
4690 4794 2
4691 4795 2
4692 4796 3
```

```
! For the items not yet handled (i.e., for data), we determine the type of
! DST record which holds the value specification and act accordingly.
```

```
CASE .DSTPTR[DST$B_TYPE] FROM 0 TO 255 OF
SET
```

```
! Handle all normal DST records, i.e. those of the standard format.
! Find the Value Spec and pass it to DBG$STA_VALSPEC for evaluation.
```

```
[DSC$K_DTYPE_LOWEST TO DSC$K_DTYPE_HIGHEST,
DST$K_BOOL, DST$K_SEPTYP, DST$K_LBLORLIT,
DST$K_ENTRY, DST$K_RTNBEG, DST$K_BLKBEQ,
DST$K_RECBEQ, DST$K_ENUMELT]:
BEGIN
```

```
! All these checks on the call to VALSPEC are here to allow the
! user to examine only registers after the completion of the user
! program. e.g. EX %R0 or EX 0\%R1
```

```
LOCAL
```

```
MODPTR : REF RST$ENTRY;
```

```
MODPTR = .SYMID[RST$L_UPSCOPEPTR];
```

```
IF (.DBG$GV_CONTROL[DBG$V_CONTROL_DONE]) AND
(.SYMID[RST$V_REGISTER]) AND
(.MODPTR NEQ 0)
```

```
THEN
```

```
IF (.MODPTR[RST$V_MCDNUMSCP]) AND
(.MODPTR[RST$L_MCDSCPNUM] EQL 0)
```

```
THEN
```

```
DBG$STA_VALSPEC(DSTPTR[DST$B_VFLAGS], .VALPTR, .VALKIND, TRUE)
```

```
ELSE
```

```
DBG$STA_VALSPEC(DSTPTR[DST$B_VFLAGS], .VALPTR, .VALKIND, FALSE)
```

```
ELSE
```

```
DBG$STA_VALSPEC(DSTPTR[DST$B_VFLAGS], .VALPTR, .VALKIND, FALSE);
```

```
END;
```

```
! Handle the Label DST record. Here we get the label address directly
! from the DST$L_VALUE field--the DST$B_VFLAGS field is not provided.
```

```
[DST$K_LABEL]:
```

```
BEGIN
```

```
VALPTR[0] = .DSTPTR[DST$L_VALUE];
```

```
VALKIND[0] = DBG$K_VAL_ADDR;
```

```
END;
```

```
! Handle the Bliss Special Cases DST record. Construct a Value Spec
! from the VFLAGS and VALUE fields in the record (which are not adjacent
! in this particular record) and call DSG$STA_VALSPEC with it.
```

```
[DST$K_BLI]:
```

```
BEGIN
```



```
4693
4694
4695
4696
4697
4698
4699
4700
4701
4702
4703
4704
4705
4706
4707
4708
4709
4710
4711
4712
4713
4714
4715
4716
4717
4718
4719
4720
4721
4722
4723
4724
4725
4726
4727
4728
4729
4730
4731
4732
4733
4734
4735
4736
4737
4738
4739
4740
4741
4742
4743
4744
4745
4746
4747
4748
4749
```

```
4797
4798
4799
4800
4801
4802
4803
4804
4805
4806
4807
4808
4809
4810
4811
4812
4813
4814
4815
4816
4817
4818
4819
4820
4821
4822
4823
4824
4825
4826
4827
4828
4829
4830
4831
4832
4833
4834
4835
4836
4837
4838
4839
4840
4841
4842
4843
4844
4845
4846
4847
4848
4849
4850
4851
4852
4853
```

LOCAL

```
MODPTR : REF RST$ENTRY,
VSPTR: REF DST$VAL_SPEC;
```

```
BLIVALSPEC[DST$B_VS_VFLAGS] = .DSTPTR[DST$B_BLI_VFLAGS];
BLITRLR = DSTPTR[DST$A_BLI_TRLR1] + .DSTPTR[DST$B_BLI_LNG];
BLIVALSPEC[DST$L_VS_VALUE] = .BLITRLR[DST$L_BLI_VALUE];
```

```
! The following is a hack to support BLISS BINDs. The
! reason for this hack is that BIND statements in BLISS
! can give rise to BLISS data whose DST type code is DST$K_BLI
! (this means they are either blocks, blockvectors, vectors,
! or bitvectors), and whose valkind is "literal". However,
! we want to treat these data items as if their valkind
! is "address". See the test TST$BLIBIND.* for an example.
! The reasons why valkind of "literal" doesn't work for these kinds
! of BLISS data are too complicated to explain here; they have to
! do with our handling of literals in general.
! So, we change valkind "literal" to valkind "address" right
! here, on the assumption that this only affects BLISS BINDs.
```

```
VSPTR = BLIVALSPEC[DST$B_VS_VFLAGS];
IF .VSPTR[DST$V_VS_VALKIND] EQL DST$K_VALKIND_LITERAL
THEN
    VSPTR[DST$V_VS_VALKIND] = DST$K_VALKIND_ADDR;
```

```
! All these checks on the call to VALSPEC are here to allow the
! user to examine only registers after the completion of the user
! program. e.g. EX $R0 or EX 0\RI
```

```
MODPTR = .SYMID[RST$L_UPSCOPEPTR];
IF (.DBG$GV_CONTROL[DBG$V_CONTROL_DONE]) AND
    (.SYMID[RST$V_REGISTER]) AND
    (.MODPTR NEQ 0)
THEN
    IF (.MODPTR[RST$V_MODNUMSCP]) AND
        (.MODPTR[RST$L_MODSCPNUM] EQL 0)
    THEN
        DBG$STA_VALSPEC(BLIVALSPEC, .VALPTR, .VALKIND, TRUE)
    ELSE
        DBG$STA_VALSPEC(BLIVALSPEC, .VALPTR, .VALKIND, FALSE)
ELSE
    DBG$STA_VALSPEC(BLIVALSPEC, .VALPTR, .VALKIND, FALSE);
END;
```

```
! Handle the Bliss field DST record. Here we just return the address of
! the number-of-components field in the DST record.
```

```
[DST$K_BLI$FLD]:
BEGIN
    VALPTR[0] = DSTPTR[DST$L_BLI$FLD_COMPS];
    VALKIND[0] = DBG$K_VAL_LITERAL;
END;
```

```

! Handle the COBOL Hack DST Record. Here we evaluate the Stack Machine
! code in the DST record and return its value as the symbol address.
!
[DST$K COB_HACK]:
    BEGIN
    CH_TRLR_PTR = DSTPTR[DST$A COBHACK_TRLR] + .DSTPTR[DST$B_NAME];
    STACK_MACHINE(CH_TRLR_PTR[DST$A_CH_STKRTN_ADDR], VALLOC, VALPTR[2])
    VALPTR[0] = .VAL[0][0];
    VALPTR[1] = 0;
    VALKIND[0] = DBG$K_VAL_ADDR;
    END;

! Handle the PSECT DST record. Here we pick the PSECT start address
! directly from the DST record.
!
[DST$K PSECT]:
    BEGIN
    VALPTR[0] = .DSTPTR[DST$L_PSECT_VALUE];
    VALKIND[0] = DBG$K_VAL_ADDR;
    END;

! Any other DST record causes an error to be signalled.
!
[INRANGE]:
    $DBG_ERROR('RSTACCESS\SYMVALUE 20');

TES;

! We have the value. Now return.
!
RETURN;

END;
```

```

56 4D 59 53 5C 53 53 45 43 43 41 54 53 52 15 00227 P.ABE: .ASCII <21>\RSTACCESS\<92>\SYMVALUE 10\
30 31 20 45 55 4C 41 00236
56 4D 59 53 5C 53 53 45 43 43 41 54 53 52 15 0023D P.ABF: .ASCII <21>\RSTACCESS\<92>\SYMVALUE 20\
30 32 20 45 55 4C 41 0024C

.PSECT DBG$CODE,NOWRT, SHR, PIC,0

007C 00000
56 00000000G 00 9E 00002 .ENTRY DBG$STA SYMVALUE, Save R2,R3,R4,R5,R6 : 4626
55 00000000G 00 9E 00009 MOVAB DBG$GV CONTROL, R6 :
5E 0C C2 00010 MOVAB LIB$SIGNAL, R5 :
53 04 AC 7D 00013 SUBL2 #12, SP :
MOVQ SYMID, R3 : 4697

```

```

B 13
16-Sep-1984 02:48:17      VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26      [DEBUG.SRC]RSTACCESS.B32;1

```

RST
V04

```
C 13
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACKACCESS.B32;1
```

Page 150
(30)

RS
VO[illegible]

RSTACCESS
V04-000

```

D 13
16-Sep-1984 02:48:17      VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26      [DEBUG.SRC]RSTACCESS.B32;1

```

Page 151
(30)

[illegible]

R

RSTACCESS
V04-000

```

E 13
16-Sep-1984 02:48:17      YAX-11 BLISS-32 V4.0-742
14-Sep-1984 12:18:26      [DEBUG.SRC]RSTACCESS.B32;1

```

Page 152
(30)

198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
78-68,-
198-68,-
198-68,-
198-68,-
198-68,-
78-68,-
78-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
78-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
78-68,-
198-68,-
168-68,-
198-68,-
198-68,-
78-68,-
198-68,-
158-68,-
178-68,-
198-68,-
78-68,-
178-68,-
198-68,-
198-68,-
78-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-
198-68,-

NS
W

49

```
F 13
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1
```

Page 153
(30)

RS
VC[illegible]

```

50      10  A3  D0 00265 78:      MOVL 16(R3), MODPTR

```

4765

		51	02	A2	9E	00269	MOVAB	2(R2), R1	4773
1C		66		06	E1	0026D	BBC	#6, DBG\$GV_CONTROL, 9\$	4766
17	15	A3		06	E1	00271	BBC	#6, 21(R3), 9\$	4767
				50	D5	00276	TSTL	MODPTR	4768
				13	13	00278	BEQL	9\$	
OE	28	A0		03	E1	0027A	BBC	#3, 40(MODPTR), 9\$	4770
			20	A0	D5	0027F	TSTL	32(MODPTR)	4771
				09	12	00282	BNEQ	9\$	
				01	DD	00284	PUSHL	#1	4773
			0C	AC	DD	00286	PUSHL	VALKIND	4775
				12	BB	00289	PUSHR	#^M<R1,R4>	
				4B	11	0028B	BRB	14\$	
				7E	D4	0028D	CLRL	-(SP)	4777
				F5	11	0028F	BRB	8\$	
	04	AE	04	A2	90	00291	MOVAB	4(DSTPTR), BLIVALSPEC	4802
		50	02	A2	9A	00296	MOVZBL	2(DSTPTR), R0	4803
		50	03	A042	9E	0029A	MOVAB	3(R0)[DSTPTR], BLITRLR	
	05	AE		60	D0	0029F	MOVL	(BLITRLR), BLIVALSPEC+1	4804
		50	04	AE	9E	002A3	MOVAB	BLIVALSPEC, VSPTR	4820
		03		60	93	002A7	BITB	(VSPTR), #3	4821
				05	12	002AA	BNEQ	11\$	
60	02	00		01	F0	002AC	INSV	#1, #0, #2, (VSPTR)	4823
		50	10	A3	D0	002B1	MOVL	16(R3), MODPTR	4830
15		66		06	E1	002B5	BBC	#6, DBG\$GV_CONTROL, 12\$	4831
10	15	A3		06	E1	002B9	BBC	#6, 21(R3), 12\$	4832
				0E	13	002BE	BEQL	12\$	4833
09	28	A0		03	E1	002C0	BBC	#3, 40(MODPTR), 12\$	4835
			20	A0	D5	002C5	TSTL	32(MODPTR)	4836
				04	12	002C8	BNEQ	12\$	
				01	DD	002CA	PUSHL	#1	4838
				02	11	002CC	BRB	13\$	4840
				7E	D4	002CE	CLRL	-(SP)	4842
			0C	AC	DD	002D0	PUSHL	VALKIND	
				54	DD	002D3	PUSHL	R4	
			10	AE	9F	002D5	PUSHAB	BLIVALSPEC	
0000V	CF			04	FB	002D8	CALLS	#4, DBG\$STA_VALSPEC	
				04	002DD	RET			4745
	64		03	A2	9E	002DE	MOVAB	3(R2), (R4)	4851
0C	BC			01	D0	002E2	MOVL	#1, @VALKIND	4852
				04	002E6	RET			4745
	50		07	A2	9A	002E7	MOVZBL	7(DSTPTR), R0	4861
	50		08	A042	9E	002EB	MOVAB	8(R0)[DSTPTR], CH_TRLR_PTR	
			08	A4	9F	002F0	PUSHAB	8(R4)	4862
			04	AE	9F	002F3	PUSHAB	VALLOC	
			01	A0	9F	002F6	PUSHAB	1(CH_TRLR_PTR)	
0000V	CF		03	FB	002F9	CALLS	#3, STACK_MACHINE		
	64		00	BE	D0	002FE	MOVL	@VALLOC, (R4)	4863
			04	A4	D4	00302	CLRL	4(R4)	4864
				04	11	00305	BRB	18\$	4865
	64		03	A2	D0	00307	MOVL	3(DSTPTR), (R4)	4874
0C	BC			02	D0	00308	MOVL	#2, @VALKIND	4875
				04	0030F	RET			4745
		00000000		EF	9F	00310	PUSHAB	P.ABF	4882
				01	DD	00316	PUSHL	#1	
		00028362		8F	DD	00318	PUSHL	#164706	
	65		03	FB	0031E	CALLS	#3, LIB\$SIGNAL		
				04	00321	RET			4891

RSTACCESS
V04-000

1 13
16-Sep-1984 02:48:17
14-Sep-1984 12:18:26

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]RSTACCESS.B32;1

Page 155
(30)

; Routine Size: 802 bytes, Routine Base: DBG\$CODE + 1E5D

```
4789 4892 1 GLOBAL ROUTINE DBG$STA_UNLOCK_SYMID(SYMID_LIST_PTR): NOVALUE =
4790 4893 1
4791 4894 1 FUNCTION
4792 4895 1 This routine "unlocks" a list of SYMIDs which have previously been
4793 4896 1 "locked" in the RST by routine DBG$STA_LOCK_SYMID. SYMIDs are locked
4794 4897 1 in the RST when the corresponding RST entries must be preserved accross
4795 4898 1 Debug commands because they are referenced by "." (current location),
4796 4899 1 breakpoints, or the like. They should then be "unlocked" when they are
4797 4900 1 no longer so referenced, i.e. when "." assumes a different value or the
4798 4901 1 breakpoint is cancelled.
4799 4902 1
4800 4903 1 The unlocking is effected by decrementing the Reference Count in the
4801 4904 1 SYMID's RST entry and all other RST entries whose reference counts were
4802 4905 1 incremented when the SYMID was originally locked. This includes all
4803 4906 1 RST entries up-scope from the original RST entry.
4804 4907 1
4805 4908 1 INPUTS
4806 4909 1 SYMID_LIST_PTR - A pointer to a linked list of Linked List Nodes, where
4807 4910 1 each node contains a forward link and a SYMID value. Each
4808 4911 1 SYMID on the list is "unlocked" in the RST by decrementing the
4809 4912 1 reference count of the corresponding RST entry.
4810 4913 1
4811 4914 1 OUTPUTS
4812 4915 1 NONE
4813 4916 1
4814 4917 1 BEGIN
4815 4918 2
4816 4919 2 LOCAL
4817 4920 2 LISTPTR: REF DBG$LINK_NODE; ! Pointer to current linked list node
4818 4921 2
4819 4922 2
4820 4923 2 ! Loop through all the SYMIDs (i.e., RST pointers) on the linked list.
4821 4924 2 ! For each SYMID on the list, call ADD_TO_REF_COUNT to decrement the RST
4822 4925 2 ! entry's reference count.
4823 4926 2
4824 4927 2 LISTPTR = .SYMID_LIST_PTR;
4825 4928 2 WHILE .LISTPTR NEQ 0 DO
4826 4929 2 BEGIN
4827 4930 2 ADD TO REF_COUNT(.LISTPTR[DBG$LINK_NODE_VALUE], -1);
4828 4931 2 LISTPTR = .LISTPTR[DBG$LINK_NODE_LINK];
4829 4932 2 END;
4830 4933 2
4831 4934 2 RETURN;
4832 4935 2
4833 4936 2
4834 4937 2
4835 4938 1 END;
```

```
52 04 0004 00000 .ENTRY DBG$STA_UNLOCK_SYMID, Save R2
AC D0 00002 MOVL SYMID_LIST_PTR, LISTPTR
10 13 00006 1$: BEQL 2$
7E 01 CE 00008 MNEGL #1, -(SP)
04 A2 DD 0000B PUSHL 4(LISTPTR)
```

```
: 4892
: 4929
: 4930
: 4932
:
```

RSTACK
V04-000

J 13
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACK.B32;1

Page 157
(31)

0000V CF
52

02 FB 0000E
62 D0 00013
EE 11 00016
04 00018 2\$:

CALLS #2, ADD TO REF COUNT
MOVL (LISTPTR), -LISTPTR
BRB 1\$
RET

:
: 4933
: 4930
: 4938

; Routine Size: 25 bytes, Routine Base: DBG\$CODE + 217F

```
4837 4939 1 GLOBAL ROUTINE DBG$STA_VALSPEC(VALSPEC, VALPTR, VALKIND, ACTUAL_REG_FLAG): NOVALUE =
4838 4940 1
4839 4941 1 FUNCTION
4840 4942 1 This routine accepts the address of a DST Value Spec as input and pro-
4841 4943 1 duces the corresponding value as output. It handles all the special
4842 4944 1 cases of value specs, including stack machine value specs, to compute
4843 4945 1 the proper output value. It requires a "context" to have been estab-
4844 4946 1 lished by DBG$STA_SETCONTEXT if there are any register references in
4845 4947 1 the value spec. If such a reference occurs and no context exists, an
4846 4948 1 error is signalled.
4847 4949 1
4848 4950 1 INPUTS
4849 4951 1 VALSPEC - A pointer to the DST Value Spec to be evaluated.
4850 4952 1
4851 4953 1 VALPTR - The address of a three-longword vector to receive the value
4852 4954 1 pointer and the corresponding stack frame pointer.
4853 4955 1
4854 4956 1 VALKIND - The address of a longword location to receive the value kind.
4855 4957 1
4856 4958 1 ACTUAL_REG_FLAG - This is a flag indicating that the valspec that we are
4857 4959 1 looking at is an actual register (e.g. %R5)
4858 4960 1
4859 4961 1 OUTPUTS
4860 4962 1 VALPTR - A pointer to the desired value is returned to VALPTR. The
4861 4963 1 byte address of the value is returned to VALPTR[0] and the
4862 4964 1 bit offset from that address is returned to VALPTR[1]. The
4863 4965 1 corresponding stack frame Pointer is returned to VALPTR[2].
4864 4966 1 VALPTR[2] will contain zero if no frame pointer is applicable.
4865 4967 1
4866 4968 1 VALKIND - The kind of the value pointed to by VALPTR is returned to
4867 4969 1 VALKIND. These are the possible values:
4868 4970 1
4869 4971 1 DBG$K_VAL_LITERAL - VALPTR points to a literal value.
4870 4972 1 DBG$K_VAL_ADDR - VALPTR contains an address.
4871 4973 1 DBG$K_VAL_DESCR - VALPTR contains the address of a
4872 4974 1 descriptor.
4873 4975 1
4874 4976 1
4875 4977 2 BEGIN
4876 4978 2
4877 4979 2 MAP
4878 4980 2 VALSPEC: REF DST$VAL_SPEC, ! Pointer to DST Value Spec to evaluate
4879 4981 2 VALPTR: REF VECTOR[3], ! Pointer to value return location
4880 4982 2 VALKIND: REF VECTOR[1]; ! Pointer to value kind return location
4881 4983 2
4882 4984 2 LOCAL
4883 4985 2 REG_FLAG,
4884 4986 2 REGNUM, ! Register number
4885 4987 2 REGPTR: REF VECTOR[.LONG], ! Pointer to register save location
4886 4988 2 VALUE: REF VECTOR[.LONG], ! Computed value
4887 4989 2 VSPTR: REF DST$VAL_SPEC; ! Pointer to current DST Value Spec
4888 4990 2
4889 4991 2 ENABLE
4890 4992 2 VALSPEC_ERROR_HANDLER; ! Set up error handler for this routine
4891 4993 2
4892 4994 2 BUILTIN
4893 4995 2 ACTUALCOUNT;
```

```
: 4894      4996      2
: 4895      4997      2
: 4896      4998      2
: 4897      4999      2
: 4898      5000      2
: 4899      5001      2
: 4900      5002      2
: 4901      5003      2
: 4902      5004      2
: 4903      5005      2
: 4904      5006      2
: 4905      5007      2
: 4906      5008      2
: 4907      5009      2
: 4908      5010      2
: 4909      5011      2
: 4910      5012      2
: 4911      5013      2
: 4912      5014      2
: 4913      5015      2
: 4914      5016      2
: 4915      5017      2
: 4916      5018      2
: 4917      5019      2
: 4918      5020      2
: 4919      5021      2
: 4920      5022      2
: 4921      5023      2
: 4922      5024      2
: 4923      5025      2
: 4924      5026      2
: 4925      5027      2
: 4926      5028      2
: 4927      5029      2
: 4928      5030      2
: 4929      5031      2
: 4930      5032      2
: 4931      5033      2
: 4932      5034      2
: 4933      5035      2
: 4934      5036      2
: 4935      5037      2
: 4936      5038      2
: 4937      5039      2
: 4938      5040      2
: 4939      5041      2
: 4940      5042      2
: 4941      5043      2
: 4942      5044      2
: 4943      5045      2
: 4944      5046      2
: 4945      5047      2
: 4946      5048      2
: 4947      5049      2
: 4948      5050      2
: 4949      5051      2
: 4950      5052      2

: Default fourth parameter to FALSE.
: IF ACTUALCOUNT() GEQ 4
: THEN
:   REG_FLAG = .ACTUAL_REG_FLAG
: ELSE
:   REG_FLAG = FALSE;

: Initially zero the returned frame pointer value in VALPTR[2]. This
: value will be changed later if a register is used in the evaluation.
: VALPTR[2] = 0;

: If the value is given by a trailing Value Spec, we get to that Value
: Spec. We loop in case the indirection is repeated.
: VSPTTR = .VALSPEC;
: WHILE .VSPTTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_TVS DO
:   VSPTTR = VSPTTR[DST$A_VS_TVS_BASE] + .VSPTTR[DST$L_VS_TVS_OFFSET];

: If the Value Spec gives the offset to a descriptor (in the DST), return
: the address of that descriptor to the caller.
: IF .VSPTTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_DSC
: THEN
:   BEGIN
:     VALPTR[0] = VSPTTR[DST$A_VS_DSC_BASE] + .VSPTTR[DST$L_VS_DSC_OFFSET];
:     VALPTR[1] = 0;
:     VALKIND[0] = DBG$K_VAL_DESCR;
:     RETURN;
:   END;

: If this is a Bit Offset Value Spec, return that bit offset as a byte
: address plus bit offset to the caller.
: IF .VSPTTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_BITOFFS
: THEN
:   BEGIN
:     VALPTR[0] = .VSPTTR[DST$L_VS_VALUE]/8;
:     VALPTR[1] = .VSPTTR[DST$L_VS_VALUE] AND 7;
:     VALKIND[0] = DBG$K_VAL_ADDR;
:     RETURN;
:   END;

: If the VFLAGS field has the special code for "unallocated", then
: put the code for "unallocated" in the kind field and then
: return. This is the case, for example, for PASCAL variables
: that are never referenced.
: IF .VSPTTR[DST$B_VS_VFLAGS] EQL DST$K_VFLAGS_UNALLOC
```

```

4951      5053      2      THEN
4952      5054      2      BEGIN
4953      5055      2      VALPTR[0] = 0;
4954      5056      2      VALPTR[1] = 0;
4955      5057      2      VALKIND[0] = DBG$K_VAL_UNALLOC;
4956      5058      2      RETURN;
4957      5059      2      END;
4958      5060
4959      5061
4960      5062      2      ! If this is a Value-Spec-follows value spec, a more complex value spec
4961      5063      2      follows the VFLAGS field. Here we handle those kinds of value specs.
4962      5064      2
4963      5065      2      IF .VSPTR[DST$B_VS_VFLAGS] EQL DST$K_VS_FOLLOWS
4964      5066      2      THEN
4965      5067      2      BEGIN
4966      5068      2
4967      5069      2      ! Sort out which particular kind of complex value specification follows.
4968      5070      2
4969      5071      2      CASE .VSPTR[DST$B_VS_ALLOC] FROM DST$K_VS_ALLOC_STAT TO DST$K_VS_ALLOC_DYN OF
4970      5072      2      SET
4971      5073      2
4972      5074      2
4973      5075      2
4974      5076      2      ! Handle statically or dynamically allocated objects without Binding
4975      5077      2      Specs. Here we just evaluate the Materialization Spec.
4976      5078      2
4977      5079      2      [DST$K_VS_ALLOC_STAT,
4978      5080      2      DST$K_VS_ALLOC_DYN]:
4979      5081      2      BEGIN
4980      5082      2      EVAL_MAT_SPEC(VSPTR[DST$A_VS_MATSPEC], .VALPTR, .VALKIND);
4981      5083      2      END;
4982      5084      2
4983      5085      2
4984      5086      2      ! Any other value in the DST$B_VS_ALLOC field is an error.
4985      5087      2
4986      5088      2      [INRANGE, OTRANGE]:
4987      5089      2      SIGNAL(DBG$ INV DSTREC);
4988      5090      2
4989      5091      2      TES;
4990      5092      2
4991      5093      2
4992      5094      2      ! We are done with the complex value spec. Return to the caller.
4993      5095      2
4994      5096      2      RETURN;
4995      5097      2      END;
4996      5098
4997      5099
4998      5100
4999      5101      2      ! This is an ordinary garden variety Value Spec with a normal VFLAGS field
5000      5102      2      and a normal VALUE field. If this is a literal, return the address of the
5001      5103      2      literal to the caller.
5002      5104      2
5003      5105      2      IF .VSPTR[DST$V_VS_VALKIND] EQL DST$K_VALKIND_LITERAL
5004      5106      2      THEN
5005      5107      2      BEGIN
5006      5108      2      VALPTR[0] = VSPTR[DST$L_VS_VALUE];
5007      5109      2      VALPTR[1] = 0;
5008      5110      2      VALKIND[0] = DBG$K_VAL_LITERAL;
```

```

: 5008      5110      3      RETURN;
: 5009      5111      2      END;
: 5010      5112      2
: 5011      5113      2
: 5012      5114      2      ! If this is a register number, return the address of the corresponding
: 5013      5115      2      ! register save area.  If the register is not available in this context,
: 5014      5116      2      ! signal an error.  (Note that we allow register 16 to mean the PSL.)
: 5015      5117      2
: 5016      5118      2      IF .VSPTR[DST$V_VS_VALKIND] EQL DST$K_VALKIND_REG
: 5017      5119      2      THEN
: 5018      5120      2          BEGIN
: 5019      5121      2              REGNUM = .VSPTR[DST$V_VS_VALUE];
: 5020      5122      2              IF (.REGNUM LSS 0) OR (.REGNUM GTR 16) THEN SIGNAL(DBG$_INV DSTREC);
: 5021      5123      2              IF .DBG$REG_VECTOR[.REGNUM] EQL 0 AND NOT .REG_FLAG
: 5022      5124      2              THEN
: 5023      5125      2                  VALSPEC_SCOPE_ERROR();
: 5024      5126      2                  VALPTR[0] = .DBG$REG_VALUES[.REGNUM];
: 5025      5127      2                  VALPTR[1] = 0;
: 5026      5128      2                  VALPTR[2] = .DBG$REG_VALUES[13];
: 5027      5129      2                  VALKIND[0] = .DBG$K_VAL_ADDR;
: 5028      5130      2                  RETURN;
: 5029      5131      2              END;
: 5030      5132      2
: 5031      5133      2
: 5032      5134      2      ! This value spec requires the value to be computed.  The resulting value is
: 5033      5135      2      ! either the address of some object or the address of a descriptor.
: 5034      5136      2
: 5035      5137      2      VALUE = .VSPTR[DST$V_VS_VALUE];
: 5036      5138      2      IF .VSPTR[DST$V_VS_DISP]
: 5037      5139      2      THEN
: 5038      5140      2          BEGIN
: 5039      5141      2              REGNUM = .VSPTR[DST$V_VS_REGNUM];
: 5040      5142      2              IF .DBG$REG_VECTOR[.REGNUM] EQL 0 THEN VALSPEC_SCOPE_ERROR();
: 5041      5143      2              VALUE = .VALUE + .DBG$REG_VALUES[.REGNUM];
: 5042      5144      2              VALPTR[2] = .DBG$REG_VALUES[13];
: 5043      5145      2              END;
: 5044      5146      2
: 5045      5147      2      IF .VSPTR[DST$V_VS_INDIRECT] THEN VALUE = .VALUE[0];
: 5046      5148      2
: 5047      5149      2
: 5048      5150      2      ! Return the computed value and its kind: address or descriptor address.
: 5049      5151      2
: 5050      5152      2      VALPTR[0] = .VALUE;
: 5051      5153      2      VALPTR[1] = 0;
: 5052      5154      2      IF .VSPTR[DST$V_VS_VALKIND] EQL DST$K_VALKIND_DESC
: 5053      5155      2      THEN
: 5054      5156      2          VALKIND[0] = .DBG$K_VAL_DESCR
: 5055      5157      2
: 5056      5158      2      ELSE
: 5057      5159      2          VALKIND[0] = .DBG$K_VAL_ADDR;
: 5058      5160      2
: 5059      5161      2      RETURN;
: 5060      5162      2
: 5061      5163      1      END;
```

					01FC 00000	.ENTRY	DBG\$STA VALSPEC, Save R2,R3,R4,R5,R6,R7,R8	4939			
				58	00000000G 00	9E 00002	MOVAB	LIB\$SIGNAL, R8			
				57	00000000G 00	9E 00009	MOVAB	DBG\$REG_VECTOR, R7			
				56	00000000G 00	9E 00010	MOVAB	DBG\$REG_VALUES, R6			
				6D	010C CF	DE 00017	MOVAL	22\$, (FP)	4977		
				04	6C	91 0001C	CMPB	(AP), #4	5000		
					06	1F 0001F	BLSSU	1\$			
				55	10 AC	D0 00021	MOVL	ACTUAL_REG_FLAG, REG_FLAG	5002		
					02	11 00025	BRB	2\$			
					55	D4 00027	CLRL	REG_FLAG	5004		
				54	08 AC	D0 00029	MOVL	VALPTR, R4	5010		
					08	A4 D4 0002D	CLRL	8(R4)			
				52	04 AC	D0 00030	MOVL	VALSPEC, VSPTR	5016		
				FB	8F	62 91 00034	CMPB	(VSPTR), #251	5017		
						0B 12 00038	BNEQ	4\$			
				50	52	01 A2	C1 0003A	ADDL3	1(VSPTR), VSPTR, R0	5018	
					52	05 A0	9E 0003F	MOVAB	5(R0), VSPTR		
						EF	11 00043	BRB	3\$		
				FA	8F	62 91 00045	CMPB	(VSPTR), #250	5024		
						0F	12 00049	BNEQ	5\$		
				50	52	01 A2	C1 0004B	ADDL3	1(VSPTR), VSPTR, R0	5027	
					64	05 A0	9E 00050	MOVAB	5(R0), (R4)		
						04	A4 D4 00054	CLRL	4(R4)	5028	
						00C3	31 00057	BRW	20\$	5029	
				FF	8F	62 91 0005A	CMPB	(VSPTR), #255	5037		
						0E	12 0005E	BNEQ	6\$		
				04	01	03 A2	C7 00060	DIVL3	#8, 1(VSPTR), (R4)	5040	
					03	00	EF 00065	EXTZV	#0, #3, 1(VSPTR), 4(R4)	5041	
						79	11 0006C	BRB	15\$	5042	
				F9	8F	62 91 0006E	CMPB	(VSPTR), #249	5052		
						07	12 00072	BNEQ	7\$		
						64	7C 00074	CLRL	(R4)	5055	
				0C	BC	04 D0 00076	MOVL	#4, @VALKIND	5057		
						04	0007A	RET		5054	
				FD	8F	62 91 0007B	CMPB	(VSPTR), #253	5065		
						21	12 0007F	BNEQ	10\$		
				01	01	03 A2	8F 00081	CASEB	3(VSPTR), #1, #1	5072	
					000E	000E	00086	.WORD	9\$-8\$, -		
									9\$-8\$		
						8F	DD 0008A	PUSHL	#164650	5089	
				68		01	FB 00090	CALLS	#1, LIB\$SIGNAL		
						04	00093	RET			
						0C	AC DD 00094	PUSHL	VALKIND	5082	
						54	DD 00097	PUSHL	R4		
						04	A2 9F 00099	PUSHAB	4(VSPTR)		
				0000V	CF	03	FB 0009C	CALLS	#3, EVAL_MAT_SPEC		
						04	000A1	RET		5067	
						62	93 000A2	BITB	(VSPTR), #3	5104	
				03		0C	12 000A5	BNEQ	11\$		
						64	01 A2	9E 000A7	MOVAB	1(R2), (R4)	5107
						04	A4 D4 000AB	CLRL	4(R4)	5108	
				0C	BC	01	D0 000AE	MOVL	#1, @VALKIND	5109	
						04	000B2	RET		5106	
				03		00	ED 000B3	CMPZV	#0, #2, (VSPTR), #3	5118	
						2F	12 000B8	BNEQ	16\$		

53	01	A2	D0	000BA	MOVL	1(VSPTR), REGNUM	5121
		05	19	000BE	BLSS	12\$	5122
10		55	D1	000C0	CMPL	REGNUM, #16	
		09	15	000C3	BLEQ	13\$	
	0002832A	8F	DD	000C5	PUSHL	#164650	
68		01	FB	000CB	CALLS	#1, LIB\$SIGNAL	
		6743	D5	000CE	TSTL	DBG\$REG_VECTOR[REGNUM]	5123
		08	12	000D1	BNEQ	14\$	
05		55	E8	000D3	BLBS	REG_FLAG, 14\$	
0000V	CF	00	FB	000D6	CALLS	#0, VALSPEC_SCOPE_ERROR	5125
64		6643	DE	000DB	MOVAL	DBG\$REG_VALUES[REGNUM], (R4)	5126
		04	A4	D4	CLRL	4(R4)	5127
08	A4	34	A6	D0	MOVL	DBG\$REG_VALUES+52, 8(R4)	5128
		39	11	000E7	BRB	21\$	5129
55		01	A2	D0	MOVL	1(VSPTR), VALUE	5137
62			03	E1	BBC	#3, (VSPTR), 18\$	5138
53	18		04	EF	EXTZV	#4, #4, (VSPTR), REGNUM	5141
62			6743	D5	TSTL	DBG\$REG_VECTOR[REGNUM]	5142
			05	12	BNEQ	17\$	
	0000V	CF	00	FB	CALLS	#0, VALSPEC_SCOPE_ERROR	
	55		6643	C0	ADDL2	DBG\$REG_VALUES[REGNUM], VALUE	5143
	08	A4	34	A6	MOVL	DBG\$REG_VALUES+52, 8(R4)	5144
		62	02	E1	BBC	#2, (VSPTR), 19\$	5147
	03	55	65	D0	MOVL	(VALUE), VALUE	
		64	55	D0	MOVL	VALUE, (R4)	5152
			04	A4	CLRL	4(R4)	5153
02	62	02	00	ED	CMPZV	#0, #2, (VSPTR), #2	5154
			05	12	BNEQ	21\$	
	0C	BC	03	D0	MOVL	#3, @VALKIND	5156
				04	RET		
	0C	BC	02	D0	MOVL	#2, @VALKIND	5159
				04	RET		5163
				0000	.WORD	Save nothing	4977
			7E	D4	CLRL	-(SP)	
			5E	DD	PUSHL	SP	
			AC	7D	MOVQ	4(AP), -(SP)	
	0000V	7E	03	FB	CALLS	#3, VALSPEC_ERROR_HANDLER	
	CF		04	00136	RET		

; Routine Size: 311 bytes, Routine Base: DBG\$CODE + 2198

```
5063 5164 1 GLOBAL ROUTINE DBG$STA_VARIANT_SELECT(TAGVALUE, VARSYMID) =
5064 5165 1
5065 5166 1 FUNCTION
5066 5167 1     This routine accepts a tag value, i.e. the value of the tag variable
5067 5168 1     in a record with variants (as in PASCAL or ADA), and a pointer to a
5068 5169 1     Variant Set RST Entry and it returns a pointer to the corresponding
5069 5170 1     variant in the Variant Set is selected by that tag value. If no
5070 5171 1     variant is selected, meaning that the tag variable has an invalid
5071 5172 1     value, a value of zero is returned. This is achieved by looping
5072 5173 1     over all the variants in the set and calling DBG$STA_VARIANT_VALUE
5073 5174 1     for each variant to determine if the tag value selects that variant.
5074 5175 1
5075 5176 1 INPUTS
5076 5177 1     TAGVALUE - The value of the tag variable to be used to select a
5077 5178 1     variant in the Variant Set. This is treated as a longword
5078 5179 1     integer value.
5079 5180 1
5080 5181 1     VARSYMID - A pointer to the Variant Set RST Entry for the Variant
5081 5182 1     Set from which a specific variant is to be selected by
5082 5183 1     TAGVALUE.
5083 5184 1
5084 5185 1 OUTPUTS
5085 5186 1     An pointer to the variant entry (obtained from the list in the Variant Set
5086 5187 1     RST Entry) is returned as the routine value. If no variant was selected
5087 5188 1     (invalid tag variable value), zero is returned.
5088 5189 1
5089 5190 1
5090 5191 2 BEGIN
5091 5192 2
5092 5193 2 MAP
5093 5194 2     VARSYMID: REF RST$ENTRY;           ! Pointer to Variant Set RST Entry
5094 5195 2
5095 5196 2 LOCAL
5096 5197 2     VARPTR: REF RST$VAR_ENTRY,         ! Pointer to current RST Variant Entry
5097 5198 2     VARSETTBL: REF VECTOR[,LONG];     ! Pointer to Variant Set RST Entry's
5098 5199 2                                         ! pointer table, where each entry
5099 5200 2                                         ! points to an RST Variant Entry
5100 5201 2
5101 5202 2
5102 5203 2
5103 5204 2 ! Check the Variant Set RST Entry pointer for validity.
5104 5205 2
5105 5206 2 IF .VARSYMID[RST$B_KIND] NEQ RST$K_VARIANT
5106 5207 2 THEN
5107 5208 2     $DBG_ERROR('RSTACCESS\VARIANT_INDEX');
5108 5209 2
5109 5210 2
5110 5211 2 ! Search through the Variant Set RST Entry's table of variants. For each
5111 5212 2 ! variant, see if TAGVALUE falls into one of its tag value ranges, and if
5112 5213 2 ! so, return the index of that variant.
5113 5214 2
5114 5215 2 VARSETTBL = VARSYMID[RST$A_VARSETTBL];
5115 5216 2 INCR I FROM 0 TO .VARSYMID[RST$L_VARSETCNT] - 1 DO
5116 5217 2     BEGIN
5117 5218 2         VARPTR = .VARSETTBL[I];
5118 5219 2         IF DBG$STA_VARIANT_VALUE(.TAGVALUE, .VARPTR[RST$L_VAR_DSTPTR])
5119 5220 2         THEN
```

```

: 5120      5221 3      RETURN .VARPTR;
: 5121      5222 2      END;
: 5122      5223 2
: 5123      5224 2
: 5124      5225 2      ! The tag value does not match the allowed tag values for any variant.
: 5125      5226 2      ! We return a value of 0 to indicate that the tag value is invalid.
: 5126      5227 2
: 5127      5228 2      RETURN 0;
: 5128      5229 2
: 5129      5230 1      END;

```

```

49 52 41 56 5C 53 53 45 43 43 41 54 53 52 17 00253 P.ABG: .PSECT DBG$PLIT,NOWRT, SHR, PIC,0
58 45 44 4E 49 5F 54 4E 41 00262 .ASCII <23>\RSTACCESS\<92>\VARIANT_INDEX\

```

```

                                .PSECT DBG$CODE,NOWRT, SHR, PIC,0
                                .ENTRY  DBG$STA VARIANT_SELECT, Save R2,R3,R4,R5
                                MOVL    VARSYMID, R4
                                CMPB    20(R4), #11
                                BEQL    1$
                                PUSHAB   P.ABG
                                PUSHL    #1
                                PUSHL    #164706
                                CALLS    #3, LIB$SIGNAL
                                MOVAB    24(R4), VARSETTBL
                                MNEGL    #1, 1
                                BRB      3$
                                MOVL     (VARSETTBL)[1], VARPTR
                                PUSHL    (VARPTR)
                                PUSHL    TAGVALUE
                                CALLS    #2, DBG$STA_VARIANT_VALUE
                                BLBC     R0, 3$
                                MOVL     VARPTR, R0
                                RET
                                AOBLS    8(R4), 1, 2$
                                CLRL     R0
                                RET
                                5164
                                5206
                                5208
                                5215
                                5219
                                5218
                                5219
                                5221
                                5216
                                5228
                                5230

```

54	08	AC	D0	00002
0B	14	A4	91	00006
		15	13	0000A
	00000000	EF	9F	0000C
		01	DD	00012
	00028362	8F	DD	00014
00000000G	00	03	FB	0001A
	52	A4	9E	00021 1\$:
	53	01	CE	00025
		15	11	00028
	55	6243	D0	0002A 2\$:
		65	DD	0002E
		AC	DD	00030
0000V	CF	02	FB	00033
	04	50	E9	00038
	50	55	D0	0003B
			04	0003E
E6	53	08	A4	F2 0003F 3\$:
			50	D4 00044
				04 00046

; Routine Size: 71 bytes, Routine Base: DBG\$CODE + 22CF

```
5131 5231 1 GLOBAL ROUTINE DBG$STA_VARIANT_VALUE(TAGVALUE, VARDSTPTR) =
5132 5232 1
5133 5233 1 FUNCTION
5134 5234 1     This routine determines whether a given tag variable value selects a
5135 5235 1     specified record variant or not. This is done by looping through all
5136 5236 1     the Tag Value Range Specifications in the variant's Variant Value DST
5137 5237 1     Record until a tag value or tag value range is found which equals or
5138 5238 1     includes the specified tag variable value. If such a match is found,
5139 5239 1     this routine returns TRUE; otherwise it returns FALSE.
5140 5240 1
5141 5241 1 INPUTS
5142 5242 1     TAGVALUE - The tag variable value. This value, treated as a longword
5143 5243 1               integer, is compared to all the tag value ranges in the
5144 5244 1               Variant Value DST Record.
5145 5245 1
5146 5246 1     VARDSTPTR - A pointer to the Variant Value DST Record for the variant
5147 5247 1               of interest. The Tag Value Range Specifications against
5148 5248 1               which TAGVALUE is checked is taken from this DST record.
5149 5249 1
5150 5250 1 OUTPUTS
5151 5251 1     If TAGVALUE selects the VARDSTPTR variant, this routine returns TRUE
5152 5252 1     as its value; otherwise FALSE is returned.
5153 5253 1
5154 5254 1
5155 5255 2 BEGIN
5156 5256 2
5157 5257 2 MAP
5158 5258 2     VARDSTPTR: REF DST$RECORD;      ! Pointer to Variant Value DST Record
5159 5259 2
5160 5260 2 LOCAL
5161 5261 2     HIGHBOND,      ! Upper bound given by the current Tag
5162 5262 2                   Value Range Specification
5163 5263 2     LOWBOUND,      ! Lower bound given by the current Tag
5164 5264 2                   Value Range Specification
5165 5265 2     RANGESPEC: REF VECTOR[.BYTE], ! Pointer to DST Tag Value Range Spec
5166 5266 2     VALKIND,        ! Value kind returned by DBG$STA_VALSPEC
5167 5267 2     VALPTR: VECTOR[3], ! Value pointer returned by STA_VALSPEC
5168 5268 2     VALSPEC: REF DST$VAL_SPEC,    ! Pointer to current DST Value Spec in
5169 5269 2                                   the current Tag Value Range Spec
5170 5270 2     VALUEPTR: REF VECTOR[1],      ! Pointer to the actual tag value given
5171 5271 2                                   by current Value Spec
5172 5272 2     VS_LENGTH;      ! Value Specification length (used to
5173 5273 2                                   find address of next Value Spec)
5174 5274 2
5175 5275 2
5176 5276 2
5177 5277 2     ! Check the Variant Value DST Record pointer for validity.
5178 5278 2
5179 5279 2 IF .VARDSTPTR[DST$B_TYPE] NEQ DST$K_VARVAL
5180 5280 2 THEN
5181 5281 2     $DBG_ERROR('RSTACCESS\VARIANT_VALUE');
5182 5282 2
5183 5283 2
5184 5284 2     ! Loop through all the Tag Value Range Specs for this particular variant.
5185 5285 2     ! If one of those values or value ranges matches the TAGVALUE parameter,
5186 5286 2     ! then we return TRUE, meaning that the specified tag value selects this
5187 5287 2     ! particular variant.
```

```
5188 5288 2
5189 5289 2
5190 5290 2
5191 5291 2
5192 5292 2
5193 5293 2
5194 5294 2
5195 5295 2
5196 5296 2
5197 5297 2
5198 5298 2
5199 5299 2
5200 5300 2
5201 5301 2
5202 5302 2
5203 5303 2
5204 5304 2
5205 5305 2
5206 5306 2
5207 5307 2
5208 5308 2
5209 5309 2
5210 5310 2
5211 5311 2
5212 5312 2
5213 5313 2
5214 5314 2
5215 5315 2
5216 5316 2
5217 5317 2
5218 5318 2
5219 5319 2
5220 5320 2
5221 5321 2
5222 5322 2
5223 5323 2
5224 5324 2
5225 5325 2
5226 5326 2
5227 5327 2
5228 5328 2
5229 5329 2
5230 5330 2
5231 5331 2
5232 5332 2
5233 5333 2
5234 5334 2
5235 5335 2
5236 5336 2
5237 5337 2
5238 5338 2
5239 5339 2
5240 5340 2
5241 5341 2
5242 5342 2
5243 5343 2
5244 5344 2

!
RANGESPEC = VARDSTPTR[DST$A_VARVAL_RNGSPEC];
INCR I FROM 0 TO .VARDSTPTR[DST$W_VARVAL_COUNT] - 1 DO
  BEGIN

    ! Pick up the first (and possibly only) value in the current Tag Value
    ! Range Specification. Then advance VALSPEC past that Value Spec.
    VALSPEC = RANGESPEC[I];
    DBG$STA VALSPEC(.VALSPEC, VALPTR, VALKIND);
    VALUEPTR = .VALPTR[0];
    LOWBOUND = .VALUEPTR[0];
    HIGHBOUND = .VALUEPTR[0];
    VS_LENGTH = 5;
    IF .VALSPEC[DST$B_VS_VFLAGS] EQL DST$K_VS_FOLLOWS
    THEN
      VS_LENGTH = .VALSPEC[DST$W_VS_LENGTH] + 3;

    VALSPEC = .VALSPEC + .VS_LENGTH;

    ! If this Tag Value Range Specification actually specifies a range,
    ! we just got the lower bound of that range. Now pick up the upper
    ! bound of the range.
    IF .RANGESPEC[0] EQL DST$K_VARVAL_RANGE
    THEN
      BEGIN
        DBG$STA VALSPEC(.VALSPEC, VALPTR, VALKIND);
        VALUEPTR = .VALPTR[0];
        HIGHBOUND = .VALUEPTR[0];
        VS_LENGTH = 5;
        IF .VALSPEC[DST$B_VS_VFLAGS] EQL DST$K_VS_FOLLOWS
        THEN
          VS_LENGTH = .VALSPEC[DST$W_VS_LENGTH] + 3;

        VALSPEC = .VALSPEC + .VS_LENGTH;
        END;

    ! See if the specified tag variable value is in the value range speci-
    ! fied by the current Tag Value Range Specification. If so, return
    ! TRUE. Otherwise, advance the RANGESPEC pointer to the next Tag Value
    ! Range Specification and loop.
    IF (.TAGVALUE GEQ .LOWBOUND) AND (.TAGVALUE LEQ .HIGHBOUND)
    THEN
      RETURN TRUE;

    RANGESPEC = .VALSPEC;
    END;

    ! End of loop over Tag Value Range Specs

    ! The specified TAGVALUE does not select this particular variant, so we
    ! return FALSE.
```

RV

```
ADDL2 VS_LENGTH, VALSPEC : 5325
```

RSTACCESS
V04-000

I 14
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 169
(34)

R
V

	59	04	AC	D1	0008A	58:	CMPL	TAGVALUE, LOWBOUND	:	5334
			0A	19	0008E		BLSS	68	:	
	57	04	AC	D1	00090		CMPL	TAGVALUE, HIGHBOUND	:	
			04	14	00094		BGTR	68	:	
	50		01	D0	00096		MOVL	#1, R0	:	5336
				04	00099		RET		:	
	53		52	D0	0009A	68:	MOVL	VALSPEC, RANGESPEC	:	5338
91	56		58	F2	0009D	78:	AOBLSS	R8, I, 28	:	5290
			50	D4	000A1		CLRL	R0	:	5345
				04	000A3		RET		:	5347

; Routine Size: 164 bytes, Routine Base: DBG\$CODE + 2316

```
: 5249      5348 1 GLOBAL ROUTINE DBG$TEST_ROUTINE_CALL( P1, P2, P3, P4 ) =
: 5250      5349 1
: 5251      5350 1 FUNCTION
: 5252      5351 1     DBG$TEST_ROUTINE_CALL is a test routine to be called from
: 5253      5352 1     the stack machine or DSI's, to test if the call to the routine
: 5254      5353 1     is correct.
: 5255      5354 1
: 5256      5355 1 INPUTS
: 5257      5356 1     P1 - first parameter
: 5258      5357 1     P2 - Second parameter
: 5259      5358 1     P3 - Third parameter
: 5260      5359 1     P4 - Fourth parameter
: 5261      5360 1
: 5262      5361 1 OUTPUTS
: 5263      5362 1     none
: 5264      5363 1
: 5265      5364 1 SIDE EFFECTS
: 5266      5365 1     none
: 5267      5366 1
: 5268      5367 2 BEGIN
: 5269      5368 2
: 5270      5369 2 RETURN P1
: 5271      5370 2
: 5272      5371 1 END;
```

```
50      04      AC      0000 00000
          7E 00002
          04 00006
```

```
.ENTRY  DBG$TEST_ROUTINE_CALL, Save nothing
MOVAB   P1, R0
RET
```

```
: 5348
: 5369
: 5371
```

; Routine Size: 7 bytes, Routine Base: DBG\$CODE + 238A


```
5274 5372 1 GLOBAL ROUTINE DBG$TRANS_TO_REGNAME (ADDRESS, NAME) =
5275 5373 1
5276 5374 1 FUNCTIONAL DESCRIPTION:
5277 5375 1
5278 5376 1 This routine determines if the input address corresponds to an address
5279 5377 1 in the context register save area. If it does, a counted string of the
5280 5378 1 register name is returned. This string includes the scope number.
5281 5379 1
5282 5380 1 FORMAL PARAMETERS:
5283 5381 1
5284 5382 1 ADDRESS - Address to be translated to a register name
5285 5383 1
5286 5384 1 NAME - The address of a longword to contain the address
5287 5385 1 of the resulting counted string.
5288 5386 1
5289 5387 1 IMPLICIT INPUTS:
5290 5388 1
5291 5389 1 DBG$REG_VALUES - Vector of context register save areas
5292 5390 1
5293 5391 1 IMPLICIT OUTPUTS:
5294 5392 1
5295 5393 1 NONE
5296 5394 1
5297 5395 1 ROUTINE VALUE:
5298 5396 1
5299 5397 1 An unsigned integer longword completion code
5300 5398 1
5301 5399 1 COMPLETION CODES:
5302 5400 1
5303 5401 1 ST$K_SUCCESS - Success. Input address mapped to register name.
5304 5402 1
5305 5403 1 ST$K_SEVERE - Failure. Input address does not correspond to
5306 5404 1 context register save area.
5307 5405 1
5308 5406 1 SIDE EFFECTS:
5309 5407 1
5310 5408 1 NONE
5311 5409 1
5312 5410 1
5313 5411 2 BEGIN
5314 5412 2
5315 5413 2 LOCAL
5316 5414 2 INDEX, ! Index into arrays
5317 5415 2 REGNAME_TABLE: VECTOR [68, LONG], ! Register name table
5318 5416 2 CONTROL_DESC: BLOCK [8, BYTE], ! $FA0 control descriptor
5319 5417 2 FAO_LENGTH: WORD, ! $FA0 output length
5320 5418 2 OUTPUT_DESC: BLOCK [8, BYTE], ! Output descriptor for FAO
5321 5419 2 OUTPUT_BUFFER: REF VECTOR [, BYTE]; ! Output buffer
5322 5420 2
5323 5421 2 BIND
5324 5422 2 FAO_STRING = UPLIT BYTE ('!UL!AC!AC'), ! $FA0 directive string
5325 5423 2 SEP_STRING = UPLIT BYTE (%ASCIC '%X'); ! Separator string
5326 5424 2
5327 5425 2
5328 5426 2
5329 5427 2 ! Fill in the register name table. Note that this MUST be done at runtime.
5330 5428 2
```

```
5331 5429 2 REGNAME-TABLE [0] = UPLIT BYTE (%ASCIC 'R0');
5332 5430 2 REGNAME-TABLE [1] = UPLIT BYTE (%ASCIC 'R0+1');
5333 5431 2 REGNAME-TABLE [2] = UPLIT BYTE (%ASCIC 'R0+2');
5334 5432 2 REGNAME-TABLE [3] = UPLIT BYTE (%ASCIC 'R0+3');
5335 5433 2 REGNAME-TABLE [4] = UPLIT BYTE (%ASCIC 'R1');
5336 5434 2 REGNAME-TABLE [5] = UPLIT BYTE (%ASCIC 'R1+1');
5337 5435 2 REGNAME-TABLE [6] = UPLIT BYTE (%ASCIC 'R1+2');
5338 5436 2 REGNAME-TABLE [7] = UPLIT BYTE (%ASCIC 'R1+3');
5339 5437 2 REGNAME-TABLE [8] = UPLIT BYTE (%ASCIC 'R2');
5340 5438 2 REGNAME-TABLE [9] = UPLIT BYTE (%ASCIC 'R2+1');
5341 5439 2 REGNAME-TABLE [10] = UPLIT BYTE (%ASCIC 'R2+2');
5342 5440 2 REGNAME-TABLE [11] = UPLIT BYTE (%ASCIC 'R2+3');
5343 5441 2 REGNAME-TABLE [12] = UPLIT BYTE (%ASCIC 'R3');
5344 5442 2 REGNAME-TABLE [13] = UPLIT BYTE (%ASCIC 'R3+1');
5345 5443 2 REGNAME-TABLE [14] = UPLIT BYTE (%ASCIC 'R3+2');
5346 5444 2 REGNAME-TABLE [15] = UPLIT BYTE (%ASCIC 'R3+3');
5347 5445 2 REGNAME-TABLE [16] = UPLIT BYTE (%ASCIC 'R4');
5348 5446 2 REGNAME-TABLE [17] = UPLIT BYTE (%ASCIC 'R4+1');
5349 5447 2 REGNAME-TABLE [18] = UPLIT BYTE (%ASCIC 'R4+2');
5350 5448 2 REGNAME-TABLE [19] = UPLIT BYTE (%ASCIC 'R4+3');
5351 5449 2 REGNAME-TABLE [20] = UPLIT BYTE (%ASCIC 'R5');
5352 5450 2 REGNAME-TABLE [21] = UPLIT BYTE (%ASCIC 'R5+1');
5353 5451 2 REGNAME-TABLE [22] = UPLIT BYTE (%ASCIC 'R5+2');
5354 5452 2 REGNAME-TABLE [23] = UPLIT BYTE (%ASCIC 'R5+3');
5355 5453 2 REGNAME-TABLE [24] = UPLIT BYTE (%ASCIC 'R6');
5356 5454 2 REGNAME-TABLE [25] = UPLIT BYTE (%ASCIC 'R6+1');
5357 5455 2 REGNAME-TABLE [26] = UPLIT BYTE (%ASCIC 'R6+2');
5358 5456 2 REGNAME-TABLE [27] = UPLIT BYTE (%ASCIC 'R6+3');
5359 5457 2 REGNAME-TABLE [28] = UPLIT BYTE (%ASCIC 'R7');
5360 5458 2 REGNAME-TABLE [29] = UPLIT BYTE (%ASCIC 'R7+1');
5361 5459 2 REGNAME-TABLE [30] = UPLIT BYTE (%ASCIC 'R7+2');
5362 5460 2 REGNAME-TABLE [31] = UPLIT BYTE (%ASCIC 'R7+3');
5363 5461 2 REGNAME-TABLE [32] = UPLIT BYTE (%ASCIC 'R8');
5364 5462 2 REGNAME-TABLE [33] = UPLIT BYTE (%ASCIC 'R8+1');
5365 5463 2 REGNAME-TABLE [34] = UPLIT BYTE (%ASCIC 'R8+2');
5366 5464 2 REGNAME-TABLE [35] = UPLIT BYTE (%ASCIC 'R8+3');
5367 5465 2 REGNAME-TABLE [36] = UPLIT BYTE (%ASCIC 'R9');
5368 5466 2 REGNAME-TABLE [37] = UPLIT BYTE (%ASCIC 'R9+1');
5369 5467 2 REGNAME-TABLE [38] = UPLIT BYTE (%ASCIC 'R9+2');
5370 5468 2 REGNAME-TABLE [39] = UPLIT BYTE (%ASCIC 'R9+3');
5371 5469 2 REGNAME-TABLE [40] = UPLIT BYTE (%ASCIC 'R10');
5372 5470 2 REGNAME-TABLE [41] = UPLIT BYTE (%ASCIC 'R10+1');
5373 5471 2 REGNAME-TABLE [42] = UPLIT BYTE (%ASCIC 'R10+2');
5374 5472 2 REGNAME-TABLE [43] = UPLIT BYTE (%ASCIC 'R10+3');
5375 5473 2 REGNAME-TABLE [44] = UPLIT BYTE (%ASCIC 'R11');
5376 5474 2 REGNAME-TABLE [45] = UPLIT BYTE (%ASCIC 'R11+1');
5377 5475 2 REGNAME-TABLE [46] = UPLIT BYTE (%ASCIC 'R11+2');
5378 5476 2 REGNAME-TABLE [47] = UPLIT BYTE (%ASCIC 'R11+3');
5379 5477 2 REGNAME-TABLE [48] = UPLIT BYTE (%ASCIC 'AP');
5380 5478 2 REGNAME-TABLE [49] = UPLIT BYTE (%ASCIC 'AP+1');
5381 5479 2 REGNAME-TABLE [50] = UPLIT BYTE (%ASCIC 'AP+2');
5382 5480 2 REGNAME-TABLE [51] = UPLIT BYTE (%ASCIC 'AP+3');
5383 5481 2 REGNAME-TABLE [52] = UPLIT BYTE (%ASCIC 'FP');
5384 5482 2 REGNAME-TABLE [53] = UPLIT BYTE (%ASCIC 'FP+1');
5385 5483 2 REGNAME-TABLE [54] = UPLIT BYTE (%ASCIC 'FP+2');
5386 5484 2 REGNAME-TABLE [55] = UPLIT BYTE (%ASCIC 'FP+3');
5387 5485 2 REGNAME-TABLE [56] = UPLIT BYTE (%ASCIC 'SP');
```

```

5388 5486 2 REGNAME_TABLE [57] = UPLIT BYTE (%ASCIC 'SP+1');
5389 5487 2 REGNAME_TABLE [58] = UPLIT BYTE (%ASCIC 'SP+2');
5390 5488 2 REGNAME_TABLE [59] = UPLIT BYTE (%ASCIC 'SP+3');
5391 5489 2 REGNAME_TABLE [60] = UPLIT BYTE (%ASCIC 'PC');
5392 5490 2 REGNAME_TABLE [61] = UPLIT BYTE (%ASCIC 'PC+1');
5393 5491 2 REGNAME_TABLE [62] = UPLIT BYTE (%ASCIC 'PC+2');
5394 5492 2 REGNAME_TABLE [63] = UPLIT BYTE (%ASCIC 'PC+3');
5395 5493 2 REGNAME_TABLE [64] = UPLIT BYTE (%ASCIC 'PSL');
5396 5494 2 REGNAME_TABLE [65] = UPLIT BYTE (%ASCIC 'PSL+1');
5397 5495 2 REGNAME_TABLE [66] = UPLIT BYTE (%ASCIC 'PSL+2');
5398 5496 2 REGNAME_TABLE [67] = UPLIT BYTE (%ASCIC 'PSL+3');
5399 5497 2
5400 5498 2
5401 5499 2 ! Check to see if the input address falls in the context register area.
5402 5500 2 ! If so, we format the scope number and register name in a buffer which
5403 5501 2 ! we then return to the caller. We return with the status STSK_SUCCESS.
5404 5502 2
5405 5503 2 IF (.ADDRESS GEQA DBG$REG_VALUES [0]) AND
5406 5504 3 (.ADDRESS LSSA DBG$REG_VALUES [17])
5407 5505 2 THEN
5408 5506 3 BEGIN
5409 5507 3
5410 5508 3 ! Calculate the register index and get a temporary memory buffer for
5411 5509 3 ! ASCII register name.
5412 5510 3
5413 5511 3 INDEX = .ADDRESS - DBG$REG_VALUES [0];
5414 5512 3 OUTPUT_BUFFER = DBG$GET_TEMPMEM(10);
5415 5513 3
5416 5514 3
5417 5515 3 ! Set up the FAO call
5418 5516 3
5419 5517 3 CONTROL_DESC [DSC$W_LENGTH] = %CHARCOUNT ('!UL!AC!AC');
5420 5518 3 CONTROL_DESC [DSC$A_POINTER] = FAO_STRING;
5421 5519 3 OUTPUT_DESC [DSC$W_LENGTH] = (10 * %UPVAL) - 1;
5422 5520 3 OUTPUT_DESC [DSC$A_POINTER] = OUTPUT_BUFFER [1];
5423 5521 3
5424 5522 3
5425 5523 3 ! Format the scope number, the separator, and the register name.
5426 5524 3
5427 5525 3 IF NOT SYSS$FAO (CONTROL_DESC,
5428 5526 3 FAO_LENGTH,
5429 5527 3 OUTPUT_DESC,
5430 5528 3 .DBG$REG_SCOPE,
5431 5529 3 SEP_STRING,
5432 5530 3 .REGNAME_TABLE [.INDEX])
5433 5531 3 THEN
5434 5532 3 $DBG_ERROR('RSTACCESS\TRANS_TO_REGNAME');
5435 5533 3
5436 5534 3
5437 5535 3 ! Copy the count into the first byte of the output buffer and return.
5438 5536 3
5439 5537 3 OUTPUT_BUFFER [0] = .FAO_LENGTH;
5440 5538 3 .NAME = .OUTPUT_BUFFER;
5441 5539 3 RETURN STSK_SUCCESS;
5442 5540 3
5443 5541 3
5444 5542 3 END
```

```
: 5445      5543      3
: 5446      5544      3
: 5447      5545      3
: 5448      5546      3
: 5449      5547      3
: 5450      5548      2
: 5451      5549      2
: 5452      5550      2
: 5453      5551      1

! The input address does not fall in the register save area. Hence we
! return the status STS$K_SEVERE to indicate this.
ELSE
    RETURN STS$K_SEVERE;
END;
```

```
.PSECT DBG$PLIT,NOWRT, SHR, PIC,0

43 41 21 43 41 21 4C 55 21 00283 P.ABI: .ASCII \!UL!AC!AC\
25 5C 02 0028C P.ABJ: .ASCII <2><92>\!
30 52 02 0028F P.ABK: .ASCII <2>\R0\
31 2B 30 52 04 00292 P.ABL: .ASCII <4>\R0+1\
32 2B 30 52 04 00297 P.ABM: .ASCII <4>\R0+2\
33 2B 30 52 04 0029C P.ABN: .ASCII <4>\R0+3\
31 2B 31 52 02 002A1 P.ABO: .ASCII <2>\R1\
31 2B 31 52 04 002A4 P.ABP: .ASCII <4>\R1+1\
32 2B 31 52 04 002A9 P.ABQ: .ASCII <4>\R1+2\
33 2B 31 52 04 002AE P.ABR: .ASCII <4>\R1+3\
32 52 02 002B3 P.ABS: .ASCII <2>\R2\
31 2B 32 52 04 002B6 P.ABT: .ASCII <4>\R2+1\
32 2B 32 52 04 002BB P.ABU: .ASCII <4>\R2+2\
33 2B 32 52 04 002C0 P.ABV: .ASCII <4>\R2+3\
33 33 52 02 002C5 P.ABw: .ASCII <2>\R3\
31 2B 33 52 04 002C8 P.ABX: .ASCII <4>\R3+1\
32 2B 33 52 04 002CD P.ABY: .ASCII <4>\R3+2\
33 2B 33 52 04 002D2 P.ABZ: .ASCII <4>\R3+3\
34 52 02 002D7 P.ACA: .ASCII <2>\R4\
31 2B 34 52 04 002DA P.ACB: .ASCII <4>\R4+1\
32 2B 34 52 04 002DF P.ACC: .ASCII <4>\R4+2\
33 2B 34 52 04 002E4 P.ACD: .ASCII <4>\R4+3\
35 52 02 002E9 P.ACE: .ASCII <2>\R5\
31 2B 35 52 04 002EC P.ACF: .ASCII <4>\R5+1\
32 2B 35 52 04 002F1 P.ACG: .ASCII <4>\R5+2\
33 2B 35 52 04 002F6 P.ACH: .ASCII <4>\R5+3\
36 52 02 002FB P.ACI: .ASCII <2>\R6\
31 2B 36 52 04 002FE P.ACJ: .ASCII <4>\R6+1\
32 2B 36 52 04 00303 P.ACK: .ASCII <4>\R6+2\
33 2B 36 52 04 00308 P.ACL: .ASCII <4>\R6+3\
37 52 02 0030D P.ACM: .ASCII <2>\R7\
31 2B 37 52 04 00310 P.ACN: .ASCII <4>\R7+1\
32 2B 37 52 04 00315 P.ACO: .ASCII <4>\R7+2\
33 2B 37 52 04 0031A P.ACP: .ASCII <4>\R7+3\
38 52 02 0031F P.ACQ: .ASCII <2>\R8\
31 2B 38 52 04 00322 P.ACR: .ASCII <4>\R8+1\
32 2B 38 52 04 00327 P.ACS: .ASCII <4>\R8+2\
33 2B 38 52 04 0032C P.ACT: .ASCII <4>\R8+3\
39 52 02 00331 P.ACU: .ASCII <2>\R9\
31 2B 39 52 04 00334 P.ACV: .ASCII <4>\R9+1\
32 2B 39 52 04 00339 P.ACW: .ASCII <4>\R9+2\
33 2B 39 52 04 0033E P.ACX: .ASCII <4>\R9+3\
30 31 52 03 00343 P.ACY: .ASCII <3>\R10\
```

4E	41	52	54	5C	53	53	45	43	43	41	4E	47	45	52	5F	4F	54	5F	53	003DC	P.AEA:	.ASCII	<26>\RSTACCESS\<92>\TRANS_TO_REGNAME\
																				003CD	P.AE:	.ASCII	<5>\R10+1\
																				00347	P.ACZ:	.ASCII	<5>\R10+2\
																				00340	P.ADA:	.ASCII	<5>\R10+3\
																				00353	P.ADB:	.ASCII	<3>\R11\
																				00359	P.ADC:	.ASCII	<5>\R11+1\
																				00350	P.ADD:	.ASCII	<5>\R11+2\
																				00363	P.ADE:	.ASCII	<5>\R11+3\
																				00369	P.ADF:	.ASCII	<2>\AP\
																				0036F	P.ADG:	.ASCII	<4>\AP+1\
																				00372	P.ADH:	.ASCII	<4>\AP+2\
																				00377	P.ADI:	.ASCII	<4>\AP+3\
																				0037C	P.ADJ:	.ASCII	<2>\FP\
																				00381	P.ADK:	.ASCII	<4>\FP+1\
																				00384	P.ADL:	.ASCII	<4>\FP+2\
																				00389	P.ADM:	.ASCII	<4>\FP+3\
																				0038E	P.ADN:	.ASCII	<2>\SP\
																				00393	P.ADO:	.ASCII	<4>\SP+1\
																				00396	P.ADP:	.ASCII	<4>\SP+2\
																				00398	P.ADQ:	.ASCII	<4>\SP+3\
																				003A0	P.ADR:	.ASCII	<2>\PC\
																				003A5	P.ADS:	.ASCII	<4>\PC+1\
																				003A8	P.ADT:	.ASCII	<4>\PC+2\
																				003AD	P.ADU:	.ASCII	<4>\PC+3\
																				003B2	P.ADV:	.ASCII	<3>\PSL\
																				003B7	P.ADW:	.ASCII	<5>\PSL+1\
																				003BB	P.ADX:	.ASCII	<5>\PSL+2\
																				003C1	P.ADY:	.ASCII	<5>\PSL+3\
																				003C7	P.ADZ:	.ASCII	<26>\RSTACCESS\<92>\TRANS_TO_REGNAME\

FAO_STRING=
SEP_STRING=P.ABI
P.ABJ

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

55	00000000G	00	9E	00002	.ENTRY	DBG\$TRANS TO REGNAME, Save R2,R3,R4,R5	5372
54	00000000'	EF	9E	00009	MOVAB	DBG\$HEG VALUES, R5	
5E	FEDC	CE	9E	00010	MOVAB	P.ABK, R4	
14	AE	64	9E	00015	MOVAB	-292(SP), SP	
18	AE	03	A4	9E	MOVAB	P.ABK, REGNAME_TABLE	5429
1C	AE	08	A4	9E	MOVAB	P.ABL, REGNAME_TABLE+4	5430
20	AE	0D	A4	9E	MOVAB	P.ABM, REGNAME_TABLE+8	5431
24	AE	12	A4	9E	MOVAB	P.ABN, REGNAME_TABLE+12	5432
28	AE	15	A4	9E	MOVAB	P.ABO, REGNAME_TABLE+16	5433
2C	AE	1A	A4	9E	MOVAB	P.ABP, REGNAME_TABLE+20	5434
30	AE	1F	A4	9E	MOVAB	P.ABQ, REGNAME_TABLE+24	5435
34	AE	24	A4	9E	MOVAB	P.ABR, REGNAME_TABLE+28	5436
38	AE	27	A4	9E	MOVAB	P.ABS, REGNAME_TABLE+32	5437
3C	AE	2C	A4	9E	MOVAB	P.ABT, REGNAME_TABLE+36	5438
40	AE	31	A4	9E	MOVAB	P.ABU, REGNAME_TABLE+40	5439
44	AE	36	A4	9E	MOVAB	P.ABV, REGNAME_TABLE+44	5440
48	AE	39	A4	9E	MOVAB	P.ABW, REGNAME_TABLE+48	5441
4C	AE	3E	A4	9E	MOVAB	P.ABX, REGNAME_TABLE+52	5442
50	AE	43	A4	9E	MOVAB	P.ABY, REGNAME_TABLE+56	5443
54	AE	48	A4	9E	MOVAB	P.ABZ, REGNAME_TABLE+60	5444
					MOVAB	P.ACA, REGNAME_TABLE+64	5445

58	AE	4B	A4	9E	00069	MOVAB	P.ACB, REGNAME+TABLE+68	5446
5C	AE	50	A4	9E	0006E	MOVAB	P.ACC, REGNAME+TABLE+72	5447
60	AE	55	A4	9E	00073	MOVAB	P.ACD, REGNAME+TABLE+76	5448
64	AE	5A	A4	9E	00078	MOVAB	P.ACE, REGNAME+TABLE+80	5449
68	AE	5D	A4	9E	0007D	MOVAB	P.ACF, REGNAME+TABLE+84	5450
6C	AE	62	A4	9E	00082	MOVAB	P.ACG, REGNAME+TABLE+88	5451
70	AE	67	A4	9E	00087	MOVAB	P.ACH, REGNAME+TABLE+92	5452
74	AE	6C	A4	9E	0008C	MOVAB	P.ACI, REGNAME+TABLE+96	5453
78	AE	6F	A4	9E	00091	MOVAB	P.ACJ, REGNAME+TABLE+100	5454
7C	AE	74	A4	9E	00096	MOVAB	P.ACK, REGNAME+TABLE+104	5455
0080	CE	79	A4	9E	0009B	MOVAB	P.ACL, REGNAME+TABLE+108	5456
0084	CE	7E	A4	9E	000A1	MOVAB	P.ACM, REGNAME+TABLE+112	5457
0088	CE	0081	C4	9E	000A7	MOVAB	P.ACN, REGNAME+TABLE+116	5458
008C	CE	0086	C4	9E	000AE	MOVAB	P.ACO, REGNAME+TABLE+120	5459
0090	CE	008B	C4	9E	000B5	MOVAB	P.ACP, REGNAME+TABLE+124	5460
FF70	CD	0090	C4	9E	000BC	MOVAB	P.ACQ, REGNAME+TABLE+128	5461
FF74	CD	0093	C4	9E	000C3	MOVAB	P.ACR, REGNAME+TABLE+132	5462
FF78	CD	0098	C4	9E	000CA	MOVAB	P.ACS, REGNAME+TABLE+136	5463
FF7C	CD	009D	C4	9E	000D1	MOVAB	P.ACT, REGNAME+TABLE+140	5464
80	AD	00A2	C4	9E	000D8	MOVAB	P.ACU, REGNAME+TABLE+144	5465
84	AD	00A5	C4	9E	000DE	MOVAB	P.ACV, REGNAME+TABLE+148	5466
88	AD	00AA	C4	9E	000E4	MOVAB	P.ACW, REGNAME+TABLE+152	5467
8C	AD	00AF	C4	9E	000EA	MOVAB	P.ACX, REGNAME+TABLE+156	5468
90	AD	00B4	C4	9E	000F0	MOVAB	P.ACY, REGNAME+TABLE+160	5469
94	AD	00B8	C4	9E	000F6	MOVAB	P.ACZ, REGNAME+TABLE+164	5470
98	AD	00BE	C4	9E	000FC	MOVAB	P.ADA, REGNAME+TABLE+168	5471
9C	AD	00C4	C4	9E	00102	MOVAB	P.ADB, REGNAME+TABLE+172	5472
A0	AD	00CA	C4	9E	00108	MOVAB	P.ADC, REGNAME+TABLE+176	5473
A4	AD	00CE	C4	9E	0010E	MOVAB	P.ADD, REGNAME+TABLE+180	5474
A8	AD	00D4	C4	9E	00114	MOVAB	P.ADE, REGNAME+TABLE+184	5475
AC	AD	00DA	C4	9E	0011A	MOVAB	P.ADF, REGNAME+TABLE+188	5476
B0	AD	00E0	C4	9E	00120	MOVAB	P.ADG, REGNAME+TABLE+192	5477
B4	AD	00E3	C4	9E	00126	MOVAB	P.ADH, REGNAME+TABLE+196	5478
B8	AD	00E8	C4	9E	0012C	MOVAB	P.ADI, REGNAME+TABLE+200	5479
BC	AD	00ED	C4	9E	00132	MOVAB	P.ADJ, REGNAME+TABLE+204	5480
C0	AD	00F2	C4	9E	00138	MOVAB	P.ADK, REGNAME+TABLE+208	5481
C4	AD	00F5	C4	9E	0013E	MOVAB	P.ADL, REGNAME+TABLE+212	5482
C8	AD	00FA	C4	9E	00144	MOVAB	P.ADM, REGNAME+TABLE+216	5483
CC	AD	00FF	C4	9E	0014A	MOVAB	P.ADN, REGNAME+TABLE+220	5484
D0	AD	0104	C4	9E	00150	MOVAB	P.ADO, REGNAME+TABLE+224	5485
D4	AD	0107	C4	9E	00156	MOVAB	P.ADP, REGNAME+TABLE+228	5486
D8	AD	010C	C4	9E	0015C	MOVAB	P.ADQ, REGNAME+TABLE+232	5487
DC	AD	0111	C4	9E	00162	MOVAB	P.ADR, REGNAME+TABLE+236	5488
E0	AD	0116	C4	9E	00168	MOVAB	P.ADS, REGNAME+TABLE+240	5489
E4	AD	0119	C4	9E	0016E	MOVAB	P.ADT, REGNAME+TABLE+244	5490
E8	AD	011E	C4	9E	00174	MOVAB	P.ADU, REGNAME+TABLE+248	5491
EC	AD	0123	C4	9E	0017A	MOVAB	P.ADV, REGNAME+TABLE+252	5492
F0	AD	0128	C4	9E	00180	MOVAB	P.ADW, REGNAME+TABLE+256	5493
F4	AD	012C	C4	9E	00186	MOVAB	P.ADX, REGNAME+TABLE+260	5494
F8	AD	0132	C4	9E	0018C	MOVAB	P.ADY, REGNAME+TABLE+264	5495
FC	AD	0138	C4	9E	00192	MOVAB	P.ADZ, REGNAME+TABLE+268	5496
	50		65	9E	00198	MOVAB	DBG\$REG_VALUES, R0	5503
	50	04	AC	D1	0019B	CMPL	ADDRESS, R0	
			6E	1F	0019F	BLSSU	2\$	
	50	44	A5	9E	001A1	MOVAB	DBG\$REG_VALUES+68, R0	5504
	50	04	AC	D1	001A5	CMPL	ADDRESS, R0	
			64	1E	001A9	BGEQU	2\$	

52	04	50	65	9E	001AB	MOVAB	DBG\$REG VALUES, R0	:	5512
		AC	50	C3	001AE	SUBL3	R0, ADDRESS, INDEX	:	
			0A	DD	001B3	PUSHL	#10	:	5513
	00000000G	00	01	FB	001B5	CALLS	#1, DBG\$GET_TEMP MEM	:	
		53	50	D0	001BC	MOVL	R0, OUTPUT_BUFFER	:	
	0C	AE	09	B0	001BF	MOVW	#9, CONTROL_DESC	:	5518
	10	AE	F4	A4	9E	001C3	MOVAB	FA0 STRING, CONTROL_DESC+4	5519
	04	AE	27	B0	001C8	MOVW	#30, OUTPUT_DESC	:	5520
	08	AE	01	A3	9E	001CC	MOVAB	1(R3), OUTPUT_DESC+4	5521
			14	AE	DD	001D1	PUSHL	REGNAME_TABLE[INDEX]	5531
			FD	A4	9F	001D5	PUSHAB	SEP_STRING	5526
		00000000	EF	DD	001D8	PUSHL	DBG\$REG SCOPE	:	5529
			10	AE	9F	001DE	PUSHAB	OUTPUT_DESC	5526
			10	AE	9F	001E1	PUSHAB	FA0_LENGTH	
			20	AE	9F	001E4	PUSHAB	CONTROL_DESC	
	00000000G	9F	06	FB	001E7	CALLS	#6, @SYS\$FA0	:	
		13	50	E8	001EE	BLBS	R0, 1\$:	
			013E	C4	9F	001F1	PUSHAB	P.AEA	5533
			01	DD	001F5	PUSHL	#1	:	
		00028362	8F	DD	001F7	PUSHL	#164706	:	
	00000000G	00	03	FB	001FD	CALLS	#3, LIB\$SIGNAL	:	
		63	6E	90	00204	1\$:	MOVW	FA0_LENGTH, (OUTPUT_BUFFER)	5538
	08	BC	53	D0	00207	MOVL	OUTPUT_BUFFER, @NAME	:	5539
		50	01	D0	0020B	MOVL	#1, R0	:	5549
				04	0020E	RET		:	
		50	04	D0	0020F	2\$:	MOVL	#4, R0	
			04	00212	RET			:	5551

; Routine Size: 531 bytes, Routine Base: DBG\$CODE + 23C1

; 5454 5552 1

```
5456 5553 1 ROUTINE ADD_TO_REF_COUNT(RSTPTR, INCREMENT): NOVALUE =
5457 5554 1
5458 5555 1 FUNCTION
5459 5556 1 This routine increments or decrements the reference count field of a
5460 5557 1 specified RST entry and all entries reachable from that entry. An RST
5461 5558 1 is "reachable" from a specified entry if it is up-scope from that entry,
5462 5559 1 if it is referenced by the RST$L_TYPEPTR field, or it is a record com-
5463 5560 1 ponent or enumeration type element of the specified Type RST Entry.
5464 5561 1
5465 5562 1 INPUTS
5466 5563 1 RSTPTR - A pointer to the RST entry whose reference count is to be
5467 5564 1 incremented or decremented.
5468 5565 1
5469 5566 1 INCREMENT - The value to be added to the RST entry's reference count.
5470 5567 1 Thus +1 increments the count and -1 decrements it.
5471 5568 1
5472 5569 1 OUTPUTS
5473 5570 1 NONE
5474 5571 1
5475 5572 1
5476 5573 2 BEGIN
5477 5574 2
5478 5575 2 MAP
5479 5576 2 RSTPTR: REF RST$ENTRY; ! Pointer to the input RST entry
5480 5577 2
5481 5578 2 LOCAL
5482 5579 2 COMPLST: REF VECTOR[,LONG], ! Pointer to Type or Variant Entry
5483 5580 2 ! component list
5484 5581 2 INVOCPTR: REF RST$ENTRY, ! Pointer to invocation number RST entry
5485 5582 2 VARPTR: REF RST$VAR_ENTRY, ! Pointer to a Variant Entry pointed to
5486 5583 2 ! by a Variant-Set RST Entry
5487 5584 2 VARSETTBL: REF VECTOR[,LONG]; ! Pointer to list of variants in a
5488 5585 2 ! Variant-Set RST Entry
5489 5586 2
5490 5587 2
5491 5588 2
5492 5589 2 ! Determine what kind of RST entry this is and act accordingly.
5493 5590 2
5494 5591 2 CASE .RSTPTR[RST$B_KIND] FROM RST$K_KIND_MINIMUM TO RST$K_KIND_MAXIMUM OF
5495 5592 2 SET
5496 5593 2
5497 5594 2
5498 5595 2 ! Handle the Module RST Entry. We increment the reference count in
5499 5596 2 case this is a "numbered scope" Module RST Entry--such entries are
5500 5597 2 created for register symbols and are on the Temporary RST Entry List.
5501 5598 2 Since a Module RST Entry terminates every up-scope chain, we return
5502 5599 2 here. This stops any up-scope recursion.
5503 5600 2
5504 5601 2 [RST$K_MODULE]:
5505 5602 3 BEGIN
5506 5603 3 RSTPTR[RST$W_REFCOUNT] = .RSTPTR[RST$W_REFCOUNT] + .INCREMENT;
5507 5604 3 RETURN;
5508 5605 2 END;
5509 5606 2
5510 5607 2
5511 5608 2 ! Handle all lexical entity and instruction label RST entries. Incre-
5512 5609 2 ment the RST entry's reference count and call ADD_TO_REF_COUNT recur-
```



```
5513 5610 2 | sively to increment reference counts in the whole up-scope chain.
5514 5611 2 |
5515 5612 2 | [RST$K_ROUTINE, RST$K_BLOCK,
5516 5613 2 | RST$K_ENTRY, RST$K_LABEL,
5517 5614 2 | RST$K_LINE, RST$K_OVERLOAD]:
5518 5615 2 | BEGIN
5519 5616 3 | RSTPTR[RST$W_REFCOUNT] = .RSTPTR[RST$W_REFCOUNT] + .INCREMENT;
5520 5617 3 | ADD_TO_REF_COUNT(.RSTPTR[RST$L_UPSCOPEPTR], .INCREMENT);
5521 5618 2 | END;
5522 5619 2 |
5523 5620 2 |
5524 5621 2 | | Handle the Data and Type Component RST Entries. Increment the refer-
5525 5622 2 | | ence count and call this routine recursively for the up-scope pointer
5526 5623 2 | | and the type pointer (if non-zero).
5527 5624 2 | |
5528 5625 2 | | [RST$K_DATA, RST$K_TYPCOMP]:
5529 5626 3 | | BEGIN
5530 5627 3 | | RSTPTR[RST$W_REFCOUNT] = .RSTPTR[RST$W_REFCOUNT] + .INCREMENT;
5531 5628 3 | | ADD_TO_REF_COUNT(.RSTPTR[RST$L_UPSCOPEPTR], .INCREMENT);
5532 5629 3 | | IF .RSTPTR[RST$L_TYPEPTR] NEQ 0
5533 5630 3 | | THEN
5534 5631 3 | | ADD_TO_REF_COUNT(.RSTPTR[RST$L_TYPEPTR], .INCREMENT);
5535 5632 2 | | END;
5536 5633 2 | |
5537 5634 2 | |
5538 5635 2 | | | Handle the Data Type RST Entry. Increment its reference count. Then
5539 5636 2 | | | call ADD_TO_REF_COUNT recursively to increment the reference counts of
5540 5637 2 | | | the up-scope chain and all record components or enumeration elements.
5541 5638 2 | | | Note that we use a mark bit to stop infinite recursion which would
5542 5639 2 | | | otherwise occur for enumeration types and possibly in other cases.
5543 5640 2 | | |
5544 5641 2 | | | [RST$K_TYPE]:
5545 5642 3 | | | BEGIN
5546 5643 3 | | | IF .RSTPTR[RST$V_MARKBIT] THEN RETURN;
5547 5644 3 | | | RSTPTR[RST$V_MARKBIT] = TRUE;
5548 5645 3 | | | RSTPTR[RST$W_REFCOUNT] = .RSTPTR[RST$W_REFCOUNT] + .INCREMENT;
5549 5646 3 | | | ADD_TO_REF_COUNT(.RSTPTR[RST$L_UPSCOPEPTR], .INCREMENT);
5550 5647 3 | | | COMPLST = RSTPTR[RST$A_TYPCOMPLST];
5551 5648 3 | | | INCR I FROM 1 TO .RSTPTR[RST$L_TYPCOMPCNT] DO
5552 5649 3 | | | ADD_TO_REF_COUNT(.COMPLST[I - 1], .INCREMENT);
5553 5650 2 | | |
5554 5651 2 | | | RSTPTR[RST$V_MARKBIT] = FALSE;
5555 5652 2 | | | END;
5556 5653 2 | | |
5557 5654 2 | | |
5558 5655 2 | | |
5559 5656 2 | | | | Handle the Variant-Set RST Entry. Here we call ADD_TO_REF_COUNT re-
5560 5657 2 | | | | cursively to cover all record components of the parent type.
5561 5658 2 | | | |
5562 5659 2 | | | | [RST$K_VARIANT]:
5563 5660 3 | | | | BEGIN
5564 5661 3 | | | | ADD_TO_REF_COUNT(.RSTPTR[RST$L_VARTAGPTR], .INCREMENT);
5565 5662 3 | | | | VARSETTBL = RSTPTR[RST$A_VARSETTBL];
5566 5663 3 | | | | INCR I FROM 1 TO .RSTPTR[RST$L_VARSETCNT] DO
5567 5664 4 | | | | BEGIN
5568 5665 4 | | | | VARPTR = .VARSETTBL[I - 1];
5569 5666 4 | | | | COMPLST = VARPTR[RST$A_VAR_COMPLST];
```

```

: 5570      5667  4      INCR J FROM 1 TO .VARPTR[RST$VAR_COMPENT] DO
: 5571      5668  4      ADD_TO_REF_COUNT(.COMPLST[J - 1], .INCREMENT);
: 5572      5669  4
: 5573      5670  3
: 5574      5671  3
: 5575      5672  2      END;
: 5576      5673  2
: 5577      5674  2
: 5578      5675  2      ! Any other kind should never show up here. If it does, error out.
: 5579      5676  2
: 5580      5677  2      [INRANGE, OUTRANGE]:
: 5581      5678  2      $DBG_ERROR('RSTACCESS\ADD_TO_REF_COUNT 10');
: 5582      5679  2
: 5583      5680  2      TES;
: 5584      5681  2
: 5585      5682  2
: 5586      5683  2      ! If there is an Invocation Number RST Entry following this one on the Sym-
: 5587      5684  2      bol chain, increment its reference count also.
: 5588      5685  2
: 5589      5686  2      IF .RSTPTR[RST$V_INVOCNUM]
: 5590      5687  2      THEN
: 5591      5688  3      BEGIN
: 5592      5689  3      INVOCPTR = .RSTPTR[RST$L_SYMCHNPTR];
: 5593      5690  3      IF .INVOCPTR[RST$B_KIND] NEQ RST$K_INVOCNUM
: 5594      5691  3      THEN
: 5595      5692  3      $DBG_ERROR('RSTACCESS\ADD_TO_REF_COUNT 20');
: 5596      5693  3
: 5597      5694  3      INVOCPTR[RST$W_REFCOUNT] = .INVOCPTR[RST$W_REFCOUNT] + .INCREMENT;
: 5598      5695  2      END;
: 5599      5696  2
: 5600      5697  2
: 5601      5698  2      ! We are all done--return.
: 5602      5699  2
: 5603      5700  2      RETURN;
: 5604      5701  2
: 5605      5702  1      END;
```

```

                                     .PSECT DBG$PLIT, NOWRT, SHR, PIC, 0
5F  44  44  41  5C  53  53  45  43  43  41  54  53  52  1D  003E8 P.AEB: .ASCII <29>\RSTACCESS\<92>\ADD_TO_REF_COUNT 1\
   31  20  54  4E  55  4F  43  5F  46  45  52  5F  4F  54  003F7
                                     .ASCII \0\
5F  44  44  41  5C  53  53  45  43  43  41  54  53  52  1D  00405 P.AEC: .ASCII <29>\RSTACCESS\<92>\ADD_TO_REF_COUNT 2\
   32  20  54  4E  55  4F  43  5F  46  45  52  5F  4F  54  00415
                                     .ASCII \0\
                                     .ASCII \0\
                                     .ASCII \0\
```

.PSECT DBG\$CODE, NOWRT, SHR, PIC, 0

07FC 00000 ADD_TO_REF_COUNT:

```

5A 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10
59          F4 AF 9E 00009 MOVAB LIB$SIGNAL, R10
56          04 AC D0 0000D MOVL  ADD_TO_REF_COUNT, R9
                    MOVL  RSTPTR, R6
```

: 5553

: 5591

0035	0035	002F	001C	0035	001C	00011	MOVAB	20(R6), R8	
0060	0042	0035	0035	0035	0035	00015	CASEB	(R8), #0, #13	
0094	0042	001C	0035	001C	00019	1\$:	.WORD	2\$-1\$,-	
		0035	0035	0035	00021			3\$-1\$,-	
					00029			4\$-1\$,-	
					00031			4\$-1\$,-	
								4\$-1\$,-	
								4\$-1\$,-	
								5\$-1\$,-	
								7\$-1\$,-	
								4\$-1\$,-	
								2\$-1\$,-	
								5\$-1\$,-	
								12\$-1\$,-	
								2\$-1\$,-	
								4\$-1\$,-	
		00000000'	EF	9F	00035	2\$:	PUSHAB	P.AEB	5678
			01	DD	0003B		PUSHL	#1	
		00028362	8F	DD	0003D		PUSHL	#164706	
	6A		03	FB	00043		CALLS	#3, LIB\$SIGNAL	
			63	11	00046		BRB	11\$	
02	A8	08	AC	A0	00048	3\$:	ADDW2	INCREMENT, 2(R8)	5603
				04	0004D		RET		5602
02	A8	08	AC	A0	0004E	4\$:	ADDW2	INCREMENT, 2(R8)	5616
		08	AC	DD	00053		PUSHL	INCREMENT	5617
		10	A6	DD	00056		PUSHL	16(R6)	
			19	11	00059		BRB	6\$	
02	A8	08	AC	A0	0005B	5\$:	ADDW2	INCREMENT, 2(R8)	5627
		08	AC	DD	00060		PUSHL	INCREMENT	5628
		10	A6	DD	00063		PUSHL	16(R6)	
	69		02	FB	00066		CALLS	#2, ADD_TO_REF_COUNT	
		18	A6	D5	00069		TSTL	24(R6)	5629
			71	13	0006C		BEQL	17\$	
		08	AC	DD	0006E		PUSHL	INCREMENT	5631
		18	A6	DD	00071		PUSHL	24(R6)	
	69		02	FB	00074	6\$:	CALLS	#2, ADD_TO_REF_COUNT	
			66	11	00077		BRB	17\$	5591
01	68		0C	E1	00079	7\$:	BBC	#12, (R8), 8\$	5644
				04	0007D		RET		
01	A8		10	88	0007E	8\$:	BISB2	#16, 1(R8)	5645
02	A8	08	AC	A0	00082		ADDW2	INCREMENT, 2(R8)	5646
		08	AC	DD	00087		PUSHL	INCREMENT	5647
		10	A6	DD	0008A		PUSHL	16(R6)	
	69		02	FB	0008D		CALLS	#2, ADD_TO_REF_COUNT	
	52	2C	A6	9E	00090		MOVAB	44(R6), -COMPLST	5648
			53	D4	00094		CLRL	I	5650
			0A	11	00096		BRB	10\$	
		08	AC	DD	00098	9\$:	PUSHL	INCREMENT	
		FC	A243	DD	0009B		PUSHL	-4(COMPLST)[I]	
	69		02	FB	0009F		CALLS	#2, ADD_TO_REF_COUNT	
F1	53	28	A6	F3	000A2	10\$:	AOBLEQ	40(R6), -I, -9\$	
	01	A8	10	8A	000A7		BICB2	#16, 1(R8)	5652
			32	11	000AB	11\$:	BRB	17\$	5591
		08	AC	DD	000AD	12\$:	PUSHL	INCREMENT	5661
		10	A6	DD	000B0		PUSHL	16(R6)	
	69		02	FB	000B3		CALLS	#2, ADD_TO_REF_COUNT	
	54	18	A6	9E	000B6		MOVAB	24(R6), -VARSETTBL	5662

RSTACCESS
V04-000

I 15
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 182
(37)

			57	D4	000BA	CLRL	I		5665
			1C	11	000BC	BRB	16\$		
	53	FC	A447	D0	000BE	13\$:	MOVL	-4(VARSETTBL)[I], VARPTR	
	52	08	A3	9E	000C3		MOVAB	8(R3), COMPLST	5666
			55	D4	000C7		CLRL	J	5668
			0A	11	000C9		BRB	15\$	
		08	AC	DD	000CB	14\$:	PUSHL	INCREMENT	
		FC	A245	DD	000CE		PUSHL	-4(COMPLST)[J]	
	69		02	FB	000D2		CALLS	#2, ADD TO REF COUNT	
F1	55	04	A3	F3	000D5	15\$:	AOBLEQ	4(VARPTR), -J, T4\$	
DF	57	08	A6	F3	000DA	16\$:	AOBLEQ	8(R6), I, 13\$	5663
20	68		0A	E1	000DF	17\$:	BBC	#10, (R8), 19\$	5686
	52	08	A6	D0	000E3		MOVL	8(R6), INVOCPTR	5689
	0C	14	A2	91	000E7		CMPB	20(INVOCPTR), #12	5690
			11	13	000EB		BEQL	18\$	
			00000000'	EF	9F 000ED		PUSHAB	P.AEC	5692
				01	DD 000F3		PUSHL	#1	
			00028362	8F	DD 000F5		PUSHL	#164706	
	6A		03	FB	000FB		CALLS	#3, LIB\$SIGNAL	
16	A2	08	AC	A0	000FE	18\$:	ADDW2	INCREMENT, 22(INVOCPTR)	5694
				04	00103	19\$:	RET		5702

; Routine Size: 260 bytes, Routine Base: DBG\$CODE + 25D4

```
: 5607 5703 1 ROUTINE CHECK_DUPLICATE(CANDLIST, INDEX1, INDEX2, ARRAY_FLAG) =
: 5608 5704 1
: 5609 5705 1 FUNCTION
: 5610 5706 1 This routine is called from the SCOPE_RULE_XXX routines to try
: 5611 5707 1 to resolve a potential ambiguity. That is, we have two candidate
: 5612 5708 1 RST entries which are in scope and appear to be equally good.
: 5613 5709 1 But before signalling 'NOUNIQUE' we may want to do some further
: 5614 5710 1 checking.
: 5615 5711 1
: 5616 5712 1 One check is for these being static data having the same address.
: 5617 5713 1 In this case, the duplicate RST entries really refer to the same
: 5618 5714 1 entity, and we can pick one arbitrarily. This situation arises
: 5619 5715 1 with FORTRAN common blocks.
: 5620 5716 1
: 5621 5717 1 Another situation where this arises is in BLISS, where the compiler
: 5622 5718 1 will put out two DST records in the same scope in situations
: 5623 5719 1 of the form:
: 5624 5720 1
: 5625 5721 1 BEGIN
: 5626 5722 1 LOCAL X;
: 5627 5723 1 ...
: 5628 5724 1 BEGIN
: 5629 5725 1 LOCAL X;
: 5630 5726 1 ...
: 5631 5727 1 END;
: 5632 5728 1 END;
: 5633 5729 1
: 5634 5730 1 Here there really is an ambiguity which the BLISS compiler should
: 5635 5731 1 resolve by putting out block-begin block-end records, but since
: 5636 5732 1 it doesn't, we arbitrarily resolve the ambiguity by picking the
: 5637 5733 1 last X. The same situation can arise with 'MAP X'.
: 5638 5734 1
: 5639 5735 1 Another situation where this arises, also in BLISS, is where the
: 5640 5736 1 same field definition occurs in many modules (perhaps because of
: 5641 5737 1 REQUIRE or LIBRARY). Instead of signalling 'NOUNIQUE' on this,
: 5642 5738 1 we check for the field definitions having identical values,
: 5643 5739 1 and if so, just return one of the RST pointers arbitrarily.
: 5644 5740 1
: 5645 5741 1 INPUTS
: 5646 5742 1 CANDBLK - A list of candidate blocks.
: 5647 5743 1
: 5648 5744 1 INDEX1 - Index into the candidate list for the first candidate.
: 5649 5745 1
: 5650 5746 1 INDEX2 - Index into the candidate list for the second candidate.
: 5651 5747 1
: 5652 5748 1 ARRAY_FLAG - If true, the symbol we are looking up was seen in a
: 5653 5749 1 subscripted expression. This may be used to resolve
: 5654 5750 1 possible ambiguities in BASIC, where it is legal to
: 5655 5751 1 have two variables of the same name, one a scalar
: 5656 5752 1 and one an array.
: 5657 5753 1
: 5658 5754 1 OUTPUTS
: 5659 5755 1 Return value is one of:
: 5660 5756 1
: 5661 5757 1 -1 : Indicates that there really is an ambiguity
: 5662 5758 1
: 5663 5759 1 one of the input parameters : means that the ambiguity was resolved
```

```

: 5664      5760 1 1      and this one was chosen.
: 5665      5761 1 1
: 5666      5762 2      BEGIN
: 5667      5763 2
: 5668      5764 2      MAP
: 5669      5765 2      CANDLST: REF VECTOR[,LONG];
: 5670      5766 2
: 5671      5767 2      LOCAL
: 5672      5768 2      ARR_FLAG,
: 5673      5769 2      BLITRLR1: REF DST$BLI_TRAILER1, ! Pointer to Bliss DST record trailer
: 5674      5770 2      BLITRLR2: REF DST$BLI_TRAILER1, ! Pointer to Bliss DST record trailer
: 5675      5771 2      CANDBLK1: REF CAND_BLOCKVECTOR,
: 5676      5772 2      CANDBLK2: REF CAND_BLOCKVECTOR,
: 5677      5773 2      COUNT1, ! Count of BLISS field values
: 5678      5774 2      COUNT2, ! Count of BLISS field values
: 5679      5775 2      DSTPTR1: REF DST$RECORD, ! Pointer to first DST record
: 5680      5776 2      DSTPTR2: REF DST$RECORD, ! Pointer to second DST record
: 5681      5777 2      FCODE1, ! fcode for first RST entry
: 5682      5778 2      FCODE2, ! fcode for second RST entry
: 5683      5779 2      PTR1, ! Pointer to BLISS field values
: 5684      5780 2      PTR2, ! Pointer to BLISS field values
: 5685      5781 2      RSTPTR1: REF RST$ENTRY, ! Pointer to first RST entry
: 5686      5782 2      RSTPTR2: REF RST$ENTRY, ! Pointer to second RST entry
: 5687      5783 2      TMPRSTPTR: REF RST$ENTRY, ! Pointer to scratch RST entry
: 5688      5784 2      TYPEID1, ! typeid for first RST entry
: 5689      5785 2      TYPEID2, ! typeid for second RST entry
: 5690      5786 2
: 5691      5787 2
: 5692      5788 2      BUILTIN
: 5693      5789 2      ACTUALCOUNT;
: 5694      5790 2
: 5695      5791 2
: 5696      5792 2
: 5697      5793 2      ! Set up the flag which says whether we are looking up a subscripted
: 5698      5794 2      ! symbol.
: 5699      5795 2
: 5700      5796 2      IF ACTUALCOUNT() GTR 3
: 5701      5797 2      THEN
: 5702      5798 2          ARR_FLAG = .ARRAY_FLAG
: 5703      5799 2
: 5704      5800 2      ELSE
: 5705      5801 2          ARR_FLAG = FALSE;
: 5706      5802 2
: 5707      5803 2
: 5708      5804 2      ! Obtain the RST entries for the two potentially duplicate symbols.
: 5709      5805 2      !
: 5710      5806 2      CANDBLK1 = .CANDLST[.INDEX1];
: 5711      5807 2      CANDBLK2 = .CANDLST[.INDEX2];
: 5712      5808 2      RSTPTR1 = .CANDBLK1[0, CAND_RSTPTR];
: 5713      5809 2      RSTPTR2 = .CANDBLK2[0, CAND_RSTPTR];
: 5714      5810 2
: 5715      5811 2
: 5716      5812 2      ! The first thing we check for is whether these are two data items
: 5717      5813 2      ! with the same static address, or two literals or enumeration
: 5718      5814 2      ! elements with the same value.
: 5719      5815 2
: 5720      5816 2      IF (.RSTPTR1[RST$B_KIND] EQL RST$K_DATA) AND
```

```

: 5721      5817 3      (.RSTPTR2[RST$B_KIND] EQL RST$K_DATA)
: 5722      5818 2      THEN
: 5723      5819 3      BEGIN
: 5724      5820 3      DSTPTR1 = .RSTPTR1[RST$L_DSTPTR];
: 5725      5821 3      DSTPTR2 = .RSTPTR2[RST$L_DSTPTR];
: 5726      5822 3
: 5727      5823 3
: 5728      5824 3      ! Check for two static data items at the same address, or two
: 5729      5825 3      ! literals or enumeration elements with the same value.
: 5730      5826 3
: 5731      5827 4      IF (((.DSTPTR1[DST$B_TYPE] GEQ DSC$K_DTYPE_LOWEST) AND (.DSTPTR1[DST$B_TYPE] LEQ DSC$K_DTYPE_HIGHEST
: 5732      5828 4          (.DSTPTR1[DST$B_TYPE] EQL DST$K_SEPTYP) OR
: 5733      5829 4          (.DSTPTR1[DST$B_TYPE] EQL DST$K_ENUMELT) OR
: 5734      5830 4          (.DSTPTR1[DST$B_TYPE] EQL DST$K_LBLORLIT))
: 5735      5831 3          AND
: 5736      5832 4          ((.DSTPTR1[DST$B_VFLAGS] EQL DST$K_VALKIND_ADDR) OR
: 5737      5833 4          (.DSTPTR1[DST$B_VFLAGS] EQL DST$K_VALKIND_LITERAL))
: 5738      5834 3      THEN
: 5739      5835 4      BEGIN
: 5740      5836 5      IF (((.DSTPTR2[DST$B_TYPE] GEQ DSC$K_DTYPE_LOWEST) AND (.DSTPTR2[DST$B_TYPE] LEQ DSC$K_DTYPE_HIG
: 5741      5837 5          (.DSTPTR2[DST$B_TYPE] EQL DST$K_SEPTYP) OR
: 5742      5838 5          (.DSTPTR2[DST$B_TYPE] EQL DST$K_ENUMELT) OR
: 5743      5839 5          (.DSTPTR2[DST$B_TYPE] EQL DST$K_LBLORLIT))
: 5744      5840 4          AND
: 5745      5841 5          ((.DSTPTR2[DST$B_VFLAGS] EQL DST$K_VALKIND_ADDR) OR
: 5746      5842 5          (.DSTPTR2[DST$B_VFLAGS] EQL DST$K_VALKIND_LITERAL))
: 5747      5843 4      THEN
: 5748      5844 5      BEGIN
: 5749      5845 5      IF (.DSTPTR1[DST$B_TYPE] EQL .DSTPTR2[DST$B_TYPE]) AND
: 5750      5846 5          (.DSTPTR1[DST$B_VFLAGS] EQL .DSTPTR2[DST$B_VFLAGS]) AND
: 5751      5847 6          (.DSTPTR1[DST$L_VALUE] EQL .DSTPTR2[DST$L_VALUE])
: 5752      5848 5      THEN
: 5753      5849 5      RETURN .INDEX1;
: 5754      5850 4      END;
: 5755      5851 3      END;
: 5756      5852 3
: 5757      5853 3
: 5758      5854 3      ! Check for two BLISS data items at the same address.
: 5759      5855 3
: 5760      5856 3      IF (.DSTPTR1[DST$B_TYPE] EQL DST$K_BLI) AND
: 5761      5857 4          (.DSTPTR2[DST$B_TYPE] EQL DST$K_BLI)
: 5762      5858 3      THEN
: 5763      5859 4      BEGIN
: 5764      5860 4      IF (.DSTPTR1[DST$B_BLI_SYM_TYPE] EQL .DSTPTR2[DST$B_BLI_SYM_TYPE]) AND
: 5765      5861 4          (.DSTPTR1[DST$B_BLI_VFLAGS] EQL DST$K_VALKIND_ADDR) AND
: 5766      5862 5          (.DSTPTR2[DST$B_BLI_VFLAGS] EQL DST$K_VALKIND_ADDR)
: 5767      5863 4      THEN
: 5768      5864 5      BEGIN
: 5769      5865 5      BLITRLR1 = DSTPTR1[DST$A_BLI_TRLR1] + .DSTPTR1[DST$B_BLI_LNG];
: 5770      5866 5      BLITRLR2 = DSTPTR2[DST$A_BLI_TRLR1] + .DSTPTR2[DST$B_BLI_LNG];
: 5771      5867 5      IF .BLITRLR1[DST$L_BLI_VALUE] EQL .BLITRLR2[DST$L_BLI_VALUE]
: 5772      5868 5      THEN
: 5773      5869 5      RETURN .INDEX1;
: 5774      5870 4      END;
: 5775      5871 3      END;
: 5776      5872 2      END;
: 5777      5873 2
```

```

5778 5874 2
5779 5875 2
5780 5876 2
5781 5877 2
5782 5878 3
5783 5879 2
5784 5880 3
5785 5881 3
5786 5882 3
5787 5883 3
5788 5884 3
5789 5885 3
5790 5886 3
5791 5887 3
5792 5888 2
5793 5889 2
5794 5890 2
5795 5891 2
5796 5892 2
5797 5893 2
5798 5894 2
5799 5895 2
5800 5896 3
5801 5897 2
5802 5898 3
5803 5899 3
5804 5900 3
5805 5901 4
5806 5902 3
5807 5903 4
5808 5904 4
5809 5905 4
5810 5906 4
5811 5907 3
5812 5908 2
5813 5909 2
5814 5910 3
5815 5911 2
5816 5912 3
5817 5913 3
5818 5914 3
5819 5915 4
5820 5916 3
5821 5917 4
5822 5918 4
5823 5919 4
5824 5920 4
5825 5921 3
5826 5922 2
5827 5923 2
5828 5924 2
5829 5925 2
5830 5926 2
5831 5927 2
5832 5928 2
5833 5929 2
5834 5930 2

! Check for two routines with the same address.
IF (.RSTPTR1[RST$B_KIND] EQL RST$K_ROUTINE) AND
  (.RSTPTR2[RST$B_KIND] EQL RST$K_ROUTINE)
THEN
  BEGIN
    IF .RSTPTR1[RST$L_STARTADDR] EQL .RSTPTR2[RST$L_STARTADDR]
    THEN
      RETURN .INDEX1;
    END;
  END;

! Check for a routine and an entry mask which are at the same
! address. This arises in PASCAL when we import a routine name
! from an environment file. We see a 'routine' DST in the module
! where the routine is really declared. We see an 'entry mask'
! DST in the module where it is imported. In this case we choose
! the routine DST.
IF (.RSTPTR1[RST$B_KIND] EQL RST$K_DATA) AND
  (.RSTPTR2[RST$B_KIND] EQL RST$K_ROUTINE)
THEN
  BEGIN
    DSTPTR1 = .RSTPTR1[RST$L_DSTPTR];
    IF (.DSTPTR1[DST$B_TYPE] EQL DSC$K_DTYPE_ZEM) AND
      (.DSTPTR1[DST$B_VFLAGS] EQL DST$K_VALRIND_ADDR)
    THEN
      BEGIN
        IF .DSTPTR1[DST$L_VALUE] EQL .RSTPTR2[RST$L_STARTADDR]
        THEN
          RETURN .INDEX2;
        END;
      END;
  END;
IF (.RSTPTR1[RST$B_KIND] EQL RST$K_ROUTINE) AND
  (.RSTPTR2[RST$B_KIND] EQL RST$K_DATA)
THEN
  BEGIN
    DSTPTR2 = .RSTPTR2[RST$L_DSTPTR];
    IF (.DSTPTR2[DST$B_TYPE] EQL DSC$K_DTYPE_ZEM) AND
      (.DSTPTR2[DST$B_VFLAGS] EQL DST$K_VALRIND_ADDR)
    THEN
      BEGIN
        IF .DSTPTR2[DST$L_VALUE] EQL .RSTPTR1[RST$L_STARTADDR]
        THEN
          RETURN .INDEX1;
        END;
      END;
  END;

! Check for language BLISS.
IMPRSTPTR = .RSTPTR1[RST$L_UPSCOPEPTR];
WHILE .IMPRSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
  IMPRSTPTR = .IMPRSTPTR[RST$L_UPSCOPEPTR];
```



```
5835 5931 2 IF .TMPRSTPTR[RST$B_LANGUAGE] EQL DBG$K_BLISS
5836 5932 2 THEN
5837 5933 2 BEGIN
5838 5934 2
5839 5935 2
5840 5936 2
5841 5937 2
5842 5938 2
5843 5939 2
5844 5940 4
5845 5941 4
5846 5942 4
5847 5943 4
5848 5944 4
5849 5945 4
5850 5946 4
5851 5947 4
5852 5948 3
5853 5949 3
5854 5950 3
5855 5951 3
5856 5952 3
5857 5953 3
5858 5954 4
5859 5955 3
5860 5956 4
5861 5957 4
5862 5958 4
5863 5959 5
5864 5960 5
5865 5961 5
5866 5962 6
5867 5963 6
5868 5964 6
5869 5965 6
5870 5966 6
5871 5967 6
5872 5968 6
5873 5969 7
5874 5970 7
5875 5971 7
5876 5972 7
5877 5973 7
5878 5974 7
5879 5975 6
5880 5976 5
5881 5977 4
5882 5978 3
5883 5979 2
5884 5980 2
5885 5981 2
5886 5982 2
5887 5983 2
5888 5984 2
5889 5985 3
5890 5986 2
5891 5987 3

: Check for duplicate data entries in BLISS.
IF .RSTPTR1[RST$L_UPSCOPEPTR] EQL .RSTPTR2[RST$L_UPSCOPEPTR]
THEN
BEGIN
: Check for duplicate data entries in BLISS.
IF .RSTPTR1[RST$L_DSTPTR] GTR .RSTPTR2[RST$L_DSTPTR]
THEN
RETURN .INDEX1
ELSE
RETURN .INDEX2;
END;

: Next, check for two occurrences of the same BLISS field.
IF (.RSTPTR1[RST$B_KIND] EQL RST$K_DATA) AND
(.RSTPTR2[RST$B_KIND] EQL RST$K_DATA)
THEN
BEGIN
IF DBG$STA_TYPEFCODE(.RSTPTR1) EQL RST$K_TYPE_BLIFLD
THEN
BEGIN
IF DBG$STA_TYPEFCODE(.RSTPTR2) EQL RST$K_TYPE_BLIFLD
THEN
BEGIN
DSTPTR1 = .RSTPTR1[RST$L_DSTPTR];
DSTPTR2 = .RSTPTR2[RST$L_DSTPTR];
COUNT1 = .DSTPTR1[DST$L_BLIFLD_COMPS];
COUNT2 = .DSTPTR2[DST$L_BLIFLD_COMPS];
IF .COUNT1 EQL .COUNT2
THEN
BEGIN
PTR1 = 1 + DSTPTR1[DST$B_NAME] + .DSTPTR1[DST$B_NAME];
PTR2 = 1 + DSTPTR2[DST$B_NAME] + .DSTPTR2[DST$B_NAME];
IF CH$EQL(.COUNT1, .PTR1, .COUNT2, .PTR2, 0)
THEN
RETURN .INDEX1;
END;
END;
END;
END;
END;

: Check for language BASIC.
IF (.TMPRSTPTR[RST$B_LANGUAGE] EQL DBG$K_BASIC) OR
(.TMPRSTPTR[RST$B_LANGUAGE] EQL DBG$K_RPG)
THEN
BEGIN
```

```
5892 5988 3 IF (.RSTPTR1[RST$B_KIND] EQL RST$K_DATA) AND
5893 5989 4 (.RSTPTR2[RST$B_KIND] EQL RST$K_DATA)
5894 5990 3 THEN
5895 5991 4 BEGIN
5896 5992 4   DBG$STA_SETCONTEXT(.RSTPTR1);
5897 5993 4   DBG$STA_SYMTYPE(.RSTPTR1, FCODE1, TYPEID1);
5898 5994 4   DBG$STA_SETCONTEXT(.RSTPTR2);
5899 5995 4   DBG$STA_SYMTYPE(.RSTPTR2, FCODE2, TYPEID2);
5900 5996 4   IF (.FCODE1 EQL RST$K_TYPE_ARRAY) AND
5901 5997 5     (.FCODE2 NEQ RST$K_TYPE_ARRAY)
5902 5998 4   THEN
5903 5999 5     BEGIN
5904 6000 5       IF .ARR_FLAG THEN RETURN .INDEX1 ELSE RETURN .INDEX2;
5905 6001 4     END;
5906 6002 4
5907 6003 4   IF (.FCODE1 NEQ RST$K_TYPE_ARRAY) AND
5908 6004 5     (.FCODE2 EQL RST$K_TYPE_ARRAY)
5909 6005 4   THEN
5910 6006 5     BEGIN
5911 6007 5       IF .ARR_FLAG THEN RETURN .INDEX2 ELSE RETURN .INDEX1;
5912 6008 4     END;
5913 6009 4
5914 6010 3   END;
5915 6011 3
5916 6012 2 END;
5917 6013 2
5918 6014 2
5919 6015 2 ! If we fall through to here then we really have an ambiguity.
5920 6016 2 ! We indicate this by returning a -1.
5921 6017 2
5922 6018 2 RETURN -1;
5923 6019 2
5924 6020 1 END;
```

```
OFFC 00000 CHECK_DUPLICATE:
5B 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 : 5703
5E          10 C2 00009 MOVAB DBG$STA_SYMTYPE, R11
03          6C 91 0000C SUBL2 #16, SP
          06 1B 0000F CMPB (AP), #3 : 5796
5A          10 AC D0 00011 BLEQU 1$
          02 11 00015 MOVL ARRAY_FLAG, ARR_FLAG : 5798
          5A D4 00017 1$: BRB 2$
57          08 AC D0 00019 2$: CLRL ARR_FLAG : 5801
51          04 BC47 D0 0001D MOVL INDEX1, R7 : 5806
59          0C AC D0 00022 MOVL @CANDLIST[R7], CANDBLK1
50          04 BC49 D0 00026 MOVL INDEX2, R9 : 5807
56          61 D0 0002B MOVL @CANDLIST[R9], CANDBLK2
55          60 D0 0002E MOVL (CANDBLK1), RSTPTR1 : 5808
          58 D4 00031 CLRL R8 : 5809
06          14 A6 91 00033 CMPB 20(RSTPTR1), #6 : 5816
          7A 12 00037 BNEQ 10$
          58 D6 00039 INCL R8
```

06	14	A5	91	0003B	CMPB	20(RSTPTR2), #6	5817	
		7F	12	0003F	BNEQ	11\$		
53	0C	A6	D0	00041	MOVL	12(RSTPTR1), DSTPTR1	5820	
52	0C	A5	D0	00045	MOVL	12(RSTPTR2), DSTPTR2	5821	
51	01	A3	9A	00049	MOVZBL	1(DSTPTR1), R1	5827	
		05	15	0004D	BLEQ	3\$		
25		51	91	0004F	CMPB	R1, #37		
		12	1B	00052	BLEQU	4\$		
A3	8F	51	91	00054	CMPB	R1, #163	5828	
		0C	13	00058	BEQL	4\$		
A4	8F	51	91	0005A	CMPB	R1, #164	5829	
		06	13	0005E	BEQL	4\$		
BA	8F	51	91	00060	CMPB	R1, #186	5830	
		46	12	00064	BNEQ	9\$		
01	02	A3	91	00066	CMPB	2(DSTPTR1), #1	5832	
		05	13	0006A	BEQL	5\$		
	02	A3	95	0006C	TSTB	2(DSTPTR1)	5833	
		3B	12	0006F	BNEQ	9\$		
50	01	A2	9A	00071	MOVZBL	1(DSTPTR2), R0	5836	
		05	15	00075	BLEQ	6\$		
25		50	91	00077	CMPB	R0, #37		
		2	1B	0007A	BLEQU	7\$		
A3	8F	50	91	0007C	CMPB	R0, #163	5837	
		0C	13	00080	BEQL	7\$		
A4	8F	50	91	00082	CMPB	R0, #164	5838	
		06	13	00086	BEQL	7\$		
BA	8F	50	91	00088	CMPB	R0, #186	5839	
		1E	12	0008C	BNEQ	9\$		
01	02	A2	91	0008E	CMPB	2(DSTPTR2), #1	5841	
		05	13	00092	BEQL	8\$		
	02	A2	95	00094	TSTB	2(DSTPTR2)	5842	
		13	12	00097	BNEQ	9\$		
50		51	D1	00099	CMPL	R1, R0	5845	
		0E	12	0009C	BNEQ	9\$		
02	A2	02	A3	91	0009E	CMPB	2(DSTPTR1), 2(DSTPTR2)	5846
		07	12	000A3	BNEQ	9\$		
03	A2	03	A3	D1	000A5	CMPL	3(DSTPTR1), 3(DSTPTR2)	5847
		48	13	000AA	BEQL	13\$		
		51	D5	000AC	TSTL	R1	5856	
		2F	12	000AE	BNEQ	12\$		
	01	A2	95	000B0	TSTB	1(DSTPTR2)	5857	
		2A	12	000B3	BNEQ	12\$		
05	A2	05	A3	91	000B5	CMPB	5(DSTPTR1), 5(DSTPTR2)	5860
		23	12	000BA	BNEQ	12\$		
01	04	A3	91	000BC	CMPB	4(DSTPTR1), #1	5861	
		1D	12	000C0	BNEQ	12\$		
01	04	A2	91	000C2	CMPB	4(DSTPTR2), #1	5862	
		17	12	000C6	BNEQ	12\$		
50	02	A3	9A	000C8	MOVZBL	2(DSTPTR1), R0	5865	
51	03	A043	9E	000CC	MOVAB	3(R0)[DSTPTR1], BLITRLR1		
50	02	A2	9A	000D1	MOVZBL	2(DSTPTR2), R0	5866	
50	03	A042	9E	000D5	MOVAB	3(R0)[DSTPTR2], BLITRLR2		
60		61	D1	000DA	CMPL	(BLITRLR1), (BLITRLR2)	5867	
		7B	13	000DD	BEQL	19\$		
		50	D4	000DF	CLRL	R0	5877	
02	14	A6	91	000E1	CMPB	20(RSTPTR1), #2		
		0F	12	000E5	BNEQ	14\$		

	02	14	50	D6	000E7	INCL	R0	:	5878
			A5	91	000E9	CMPB	20(RSTPTR2), #2	:	
			07	12	000ED	BNEQ	14\$:	
18	A5	18	A6	D1	000EF	CMPL	24(RSTPTR1), 24(RSTPTR2)	:	5881
			64	13	000F4	BEQL	19\$:	
	1D		58	E9	000F6	BLBC	R8, 15\$:	5895
	02	14	A5	91	000F9	CMPB	20(RSTPTR2), #2	:	5896
			17	12	000FD	BNEQ	15\$:	
	53	0C	A6	D0	000FF	MOVL	12(RSTPTR1), DSTPTR1	:	5899
	17	01	A3	91	00103	CMPB	1(DSTPTR1), #23	:	5900
			0D	12	00107	BNEQ	15\$:	
	01	02	A3	91	00109	CMPB	2(DSTPTR1), #1	:	5901
			07	12	0010D	BNEQ	15\$:	
18	A5	03	A3	D1	0010F	CMPL	3(DSTPTR1), 24(RSTPTR2)	:	5904
			47	13	00114	BEQL	20\$:	
	1D		50	E9	00116	BLBC	R0, 16\$:	5909
	06	14	A5	91	00119	CMPB	20(RSTPTR2), #6	:	5910
			17	12	0011D	BNEQ	16\$:	
	52	0C	A5	D0	0011F	MOVL	12(RSTPTR2), DSTPTR2	:	5913
	17	01	A2	91	00123	CMPB	1(DSTPTR2), #23	:	5914
			0D	12	00127	BNEQ	16\$:	
	01	02	A2	91	00129	CMPB	2(DSTPTR2), #1	:	5915
			07	12	0012D	BNEQ	16\$:	
18	A6	03	A2	D1	0012F	CMPL	3(DSTPTR2), 24(RSTPTR1)	:	5918
			24	13	00134	BEQL	19\$:	
	54	10	A6	D0	00136	MOVL	16(RSTPTR1), TMPRSTPTR	:	5927
	01	14	A4	91	0013A	CMPB	20(TMPRSTPTR), #1	:	5928
			06	13	0013E	BEQL	18\$:	
	54	10	A4	D0	00140	MOVL	16(TMPRSTPTR), TMPRSTPTR	:	5929
			F4	11	00144	BRB	17\$:	
	02	29	A4	91	00146	CMPB	41(TMPRSTPTR), #2	:	5931
			68	12	0014A	BNEQ	22\$:	
10	A5	10	A6	D1	0014C	CMPL	16(RSTPTR1), 16(RSTPTR2)	:	5938
			0D	12	00151	BNEQ	21\$:	
0C	A5	0C	A6	D1	00153	CMPL	12(RSTPTR1), 12(RSTPTR2)	:	5941
			03	15	00158	BLEQ	20\$:	
			00B6	31	0015A	BRW	26\$:	
			00AF	31	0015D	BRW	25\$:	
	51		58	E9	00160	BLBC	R8, 22\$:	5953
	06	14	A5	91	00163	CMPB	20(RSTPTR2), #6	:	5954
			4B	12	00167	BNEQ	22\$:	
			56	DD	00169	PUSHL	RSTPTR1	:	5957
00000000G	00		01	FB	0016B	CALLS	#1, DBG\$STA_TYPEFCODE	:	
	0E		50	D1	00172	CMPL	R0, #14	:	
			3D	12	00175	BNEQ	22\$:	
			55	DD	00177	PUSHL	RSTPTR2	:	5960
00000000G	00		01	FB	00179	CALLS	#1, DBG\$STA_TYPEFCODE	:	
	0E		50	D1	00180	CMPL	R0, #14	:	
			2F	12	00183	BNEQ	22\$:	
	53	0C	A6	D0	00185	MOVL	12(RSTPTR1), DSTPTR1	:	5963
	52	0C	A5	D0	00189	MOVL	12(RSTPTR2), DSTPTR2	:	5964
	58	03	A3	D0	0018D	MOVL	3(DSTPTR1), COUNT1	:	5965
	51	03	A2	D0	00191	MOVL	3(DSTPTR2), COUNT2	:	5966
	51		58	D1	00195	CMPL	COUNT1, COUNT2	:	5967
			1A	12	00198	BNEQ	22\$:	
	50	07	A3	9A	0019A	MOVZBL	7(DSTPTR1), R0	:	5970
	50	08	A043	9E	0019E	MOVAB	8(R0)[DSTPTR1], PTR1	:	

51

00

53	07	A2	9A	001A3	MOVZBL	7(DSTPTR2), R3	5971	
52	08	A3	42	9E	001A7	MOVAB	8(R3)[DSTPTR2], PTR2	5972
60		58	2D	001AC	CMPC5	COUNT1, (PTR1), #0, COUNT2, (PTR2)		
		62		001B1				
		5F	13	001B2	BEQL	26\$		
04	29	A4	91	001B4	22\$: CMPB	41(IMPRSTPTR), #4	5984	
		06	13	001B8	BEQL	23\$		
08	29	A4	91	001BA	CMPB	41(IMPRSTPTR), #8	5985	
		57	12	001BE	BNEQ	27\$		
06	14	A6	91	001C0	23\$: CMPB	20(RSTPTR1), #6	5988	
		51	12	001C4	BNEQ	27\$		
06	14	A5	91	001C6	CMPB	20(RSTPTR2), #6	5989	
		4B	12	001CA	BNEQ	27\$		
		56	DD	001CC	PUSHL	RSTPTR1	5992	
EEE4	CF	01	FB	001CE	CALLS	#1, DBG\$STA_SETCONTEXT		
		5E	DD	001D3	PUSHL	SP	5993	
		08	AE	9F	001D5	PUSHAB	FCODE1	
		56	DD	001D8	PUSHL	RSTPTR1		
6B		03	FB	001DA	CALLS	#3, DBG\$STA_SYMTYPE		
		55	DD	001DD	PUSHL	RSTPTR2	5994	
EED3	CF	01	FB	001DF	CALLS	#1, DBG\$STA_SETCONTEXT		
		08	AE	9F	001E4	PUSHAB	TYPEID2	5995
		10	AE	9F	001E7	PUSHAB	FCODE2	
		55	DD	001EA	PUSHL	RSTPTR2		
6B		03	FB	001EC	CALLS	#3, DBG\$STA_SYMTYPE		
01	04	AE	D1	001EF	CMPL	FCODE1, #1	5996	
		0B	12	001F3	BNEQ	24\$		
01	0C	AE	D1	001F5	CMPL	FCODE2, #1	5997	
		05	13	001F9	BEQL	24\$		
11		5A	E9	001FB	BLBC	ARR_FLAG, 25\$	6000	
		13	11	001FE	BRB	26\$		
01	04	AE	D1	00200	24\$: CMPL	FCODE1, #1	6003	
		11	13	00204	BEQL	27\$		
01	0C	AE	D1	00206	CMPL	FCODE2, #1	6004	
		0B	12	0020A	BNEQ	27\$		
04		5A	E9	0020C	BLBC	ARR_FLAG, 26\$	6007	
50		59	D0	0020F	25\$: MOVL	R9, R0		
		04	00212	RET				
50		57	D0	00213	26\$: MOVL	R7, R0		
		04	00216	RET				
50		01	CE	00217	27\$: MNEGL	#1, R0	6018	
		04	0021A	RET			6020	

; Routine Size: 539 bytes, Routine Base: DBG\$CODE + 26D8

```
5926 6021 1 ROUTINE EVAL_MAT_SPEC(MSPTR, VALPTR, VALKIND): NOVALUE =
5927 6022 1
5928 6023 1 FUNCTION
5929 6024 1 This routine evaluates a Materialization Spec. Materialization Specs
5930 6025 1 are found inside certain kinds of Value Specs when more complex computa-
5931 6026 1 tions are needed to produce a symbol's value. In particular, they are
5932 6027 1 used when a call on a compiler-provided run-time routine or an invoca-
5933 6028 1 tion of the DST stack machine is used to compute the value.
5934 6029 1
5935 6030 1 INPUTS
5936 6031 1 MSPTR - Pointer to the Materialization Spec to be evaluated.
5937 6032 1
5938 6033 1 VALPTR - The address of a three-longword vector to receive the value
5939 6034 1 pointer and the corresponding stack frame pointer.
5940 6035 1
5941 6036 1 VALKIND - The address of a longword location to receive the value kind.
5942 6037 1
5943 6038 1 OUTPUTS
5944 6039 1 VALPTR - A pointer to the desired value is returned to VALPTR. The
5945 6040 1 byte address of the value is returned to VALPTR[0] and the
5946 6041 1 bit offset from that address is returned to VALPTR[1]. The
5947 6042 1 corresponding stack Frame Pointer is returned to VALPTR[2].
5948 6043 1 VALPTR[2] will contain zero if no frame pointer is applicable.
5949 6044 1
5950 6045 1 VALKIND - The kind of the value pointed to by VALPTR is returned to
5951 6046 1 VALKIND. These are the possible values:
5952 6047 1
5953 6048 1 DBG$K_VAL_LITERAL - VALPTR points to a literal value.
5954 6049 1 DBG$K_VAL_ADDR - VALPTR contains an address.
5955 6050 1 DBG$K_VAL_DESCR - VALPTR contains the address of a
5956 6051 1 descriptor.
5957 6052 1
5958 6053 1 No value is returned by EVAL_MAT_SPEC.
5959 6054 1
5960 6055 1
5961 6056 2 BEGIN
5962 6057 2
5963 6058 2 MAP
5964 6059 2 MSPTR: REF DST$MATER_SPEC, ! Pointer to the materialization spec
5965 6060 2 VALPTR: REF VECTOR[3], ! Pointer to value pointer vector
5966 6061 2 VALKIND: REF VECTOR[1]; ! Pointer to value kind location
5967 6062 2
5968 6063 2 LOCAL
5969 6064 2 VALLOC: REF VECTOR[.LONG], ! Pointer to value as computed by the
5970 6065 2 ! mechanism specified in the spec
5971 6066 2 REGNUM; ! Register number
5972 6067 2
5973 6068 2
5974 6069 2
5975 6070 2 ! Determine what kind of materialization spec we have. Compute the value
5976 6071 2 ! of each kind as appropriate.
5977 6072 2
5978 6073 2 CASE .MSPTR[DST$B_MS_MECH] FROM DST$K_MS_MECH_MIN TO DST$K_MS_MECH_MAX OF
5979 6074 2 SET
5980 6075 2
5981 6076 2
5982 6077 2 ! Routine Call mechanism spec. Call a run-time routine provided by the
```

```

: 5983      6078 2      ! compiler in the user image to compute the desired value.
: 5984      6079 2
: 5985      6080 2
: 5986      6081 2
: 5987      6082 2
: 5988      6083 2
: 5989      6084 2
: 5990      6085 2
: 5991      6086 2
: 5992      6087 2
: 5993      6088 2
: 5994      6089 2
: 5995      6090 2
: 5996      6091 2
: 5997      6092 2
: 5998      6093 2
: 5999      6094 2
: 6000      6095 2
: 6001      6096 2
: 6002      6097 2
: 6003      6098 2
: 6004      6099 2
: 6005      6100 2
: 6006      6101 2
: 6007      6102 2
: 6008      6103 2
: 6009      6104 2
: 6010      6105 2
: 6011      6106 2
: 6012      6107 2
: 6013      6108 2
: 6014      6109 2
: 6015      6110 2
: 6016      6111 2
: 6017      6112 2
: 6018      6113 2
: 6019      6114 2
: 6020      6115 2
: 6021      6116 2
: 6022      6117 2
: 6023      6118 2
: 6024      6119 2
: 6025      6120 2
: 6026      6121 2
: 6027      6122 2
: 6028      6123 2
: 6029      6124 2
: 6030      6125 2
: 6031      6126 2
: 6032      6127 2
: 6033      6128 2
: 6034      6129 2
: 6035      6130 2
: 6036      6131 2
: 6037      6132 2
: 6038      6133 2
: 6039      6134 2

! compiler in the user image to compute the desired value.
[DST$K_MS_MECH_RTNCALL]:
  BEGIN
    VALPTR[2] = .DBG$REG.VALUES[13];
    VALLOC = DBG$GET_TEMPMEM(4);
    VALSPEC_ROUT_CALL(.VALLOC, .MSPTR[DST$L_MS_MECH_RTNADDR],
                      .MSPTR[DST$V_MS_DUMARG], TRUE);
  END;

! Routine Call mechanism spec. Call a run-time routine provided by the
! compiler in the user image to compute the desired value. This differs
! from the above RTNCALL case only in that the FP is not passed into
! the thunk. The last parameter to VALSPEC_ROUT_CALL is a flag indicating
! this.
[DST$K_MS_MECH_RTN_NOFP]:
  BEGIN
    VALPTR[2] = .DBG$REG.VALUES[13];
    VALLOC = DBG$GET_TEMPMEM(4);
    VALSPEC_ROUT_CALL(.VALLOC, .MSPTR[DST$L_MS_MECH_RTNADDR],
                      .MSPTR[DST$V_MS_DUMARG], FALSE);
  END;

! Stack machine mechanism spec. Use the DST 'stack machine' to compute
! the desired value.
[DST$K_MS_MECH_STK]:
  STACK_MACHINE(MSPTR[DST$A_MS_MECH_SPEC], VALLOC, VALPTR[2]);

! Any other value is an error.
[OUTRANGE]:
  SIGNAL(DBG$_INVDSTREC);

TES;

! We have now computed the value. Return the value and the value kind to
! the caller.
CASE .MSPTR[DST$B_MS_KIND] FROM DST$K_MS_BYTADDR TO DST$K_MS_DSC OF
  SET

! We have a byte address.
[DST$K_MS_BYTADDR]:
  BEGIN
    VALPTR[0] = .VALLOC[0];
    VALPTR[1] = 0;
    VALKIND[0] = DBG$K_VAL_ADDR;
  END;
```

```

: 6040      6135 2
: 6041      6136 2
: 6042      6137 2
: 6043      6138 2
: 6044      6139 2
: 6045      6140 2
: 6046      6141 2
: 6047      6142 2
: 6048      6143 2
: 6049      6144 2
: 6050      6145 2
: 6051      6146 2
: 6052      6147 2
: 6053      6148 2
: 6054      6149 2
: 6055      6150 2
: 6056      6151 2
: 6057      6152 2
: 6058      6153 2
: 6059      6154 2
: 6060      6155 2
: 6061      6156 2
: 6062      6157 2
: 6063      6158 2
: 6064      6159 2
: 6065      6160 2
: 6066      6161 2
: 6067      6162 2
: 6068      6163 2
: 6069      6164 2
: 6070      6165 2
: 6071      6166 2
: 6072      6167 2
: 6073      6168 2
: 6074      6169 2
: 6075      6170 2
: 6076      6171 2
: 6077      6172 2
: 6078      6173 2
: 6079      6174 2
: 6080      6175 2
: 6081      6176 2
: 6082      6177 2
: 6083      6178 2
: 6084      6179 2
: 6085      6180 2
: 6086      6181 2
: 6087      6182 2
: 6088      6183 2
: 6089      6184 2
: 6090      6185 2
: 6091      6186 2
: 6092      6187 2
: 6093      6188 2
: 6094      6189 2
: 6095      6190 2
: 6096      6191 2

! We have a bit address, i.e. a byte address with a bit offset.
[DST$K_MS_BITADDR]:
  BEGIN
    VALPTR[0] = .VALLOC[0];
    VALPTR[1] = .VALLOC[1];
    VALKIND[0] = DBG$K_VAL_ADDR;
  END;

! We have a bit offset.
[DST$K_MS_BITOFFS]:
  BEGIN
    VALPTR[0] = .VALLOC[0]/8;
    VALPTR[1] = .VALLOC[0] AND 7;
    VALKIND[0] = DBG$K_VAL_ADDR;
  END;

! We have an "R-value", i.e a literal or constant value.
[DST$K_MS_RVAL]:
  BEGIN
    VALPTR[0] = VALLOC[0];
    VALPTR[1] = 0;
    VALKIND[0] = DBG$K_VAL_LITERAL;
  END;

! We have a register number. Convert it into a pointer into the regis-
! ter value vector.
[DST$K_MS_REG]:
  BEGIN
    REGNUM = .VALLOC[0];
    IF (.REGNUM LSS 0) OR (.REGNUM GTR 15) THEN SIGNAL(DBG$ INVDSITREC);
    IF .DBG$REG_VECTOR[.REGNUM] EQL 0 THEN VALSPEC_SCOPE_ERROR();
    VALPTR[0] = DBG$REG_VALUES[.REGNUM];
    VALPTR[1] = 0;
    VALPTR[2] = .DBG$REG_VALUES[13];
    VALKIND[0] = DBG$K_VAL_ADDR;
  END;

! We have a descriptor address.
[DST$K_MS_DSC]:
  BEGIN
    VALPTR[0] = VALLOC[0];
    VALPTR[1] = 0;
    VALKIND[0] = DBG$K_VAL_DESCR;
  END;

! Any other value is an error.
```


Page 195
(39)

003C 00000 EVAL_MAT_SPEC:

PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

; Routine Size: 279 bytes, Routine Base: DBG\$CODE + 28F3

```

6111 6204 1 ROUTINE FOLLOW_STATIC_LINK(RSTPTR, SCOPE_RSTPTR) =
6112 6205 1
6113 6206 1 FUNCTION
6114 6207 1 This routine determines the proper invocation number for a data item
6115 6208 1 which has been looked up in a specific scope. This is accomplished by
6116 6209 1 starting with the call frame of the routine defining the scope and then
6117 6210 1 following the Static Links (in the call stack) until we get to a frame
6118 6211 1 for the routine in which the data item is declared. The invocation
6119 6212 1 number of that routine is computed along the way, and is returned as the
6120 6213 1 invocation number of the data item.
6121 6214 1
6122 6215 1 The Static Links take us from call frame to call frame as we go up-scope
6123 6216 1 from the scope routine to the declaring routine. Static links are spec-
6124 6217 1 ified by Value Specs in Static Link DST records. There can be one such
6125 6218 1 record per routine. However, if no such record is specified (BLISS, for
6126 6219 1 example, does not use them), we take the first invocation we find in the
6127 6220 1 call stack (after the current call frame) for the up-scope routine. The
6128 6221 1 Static Link DST record always gives the right final invocation number,
6129 6222 1 but the first-invocation-we-find method works equally well in all but a
6130 6223 1 few anomalous cases.
6131 6224 1
6132 6225 1 INPUTS
6133 6226 1 RSTPTR - Pointer to the RST entry of the object (normally a data item)
6134 6227 1 whose invocation number is to be determined.
6135 6228 1
6136 6229 1 SCOPE_RSTPTR - Pointer to the RST entry which defines the scope in which
6137 6230 1 the RSTPTR item is to be found. This scope defines the invoc-
6138 6231 1 ation of RSTPTR we want. This routine assumes that the RSTPTR
6139 6232 1 item is known to be in the scope defined by SCOPE_RSTPTR.
6140 6233 1
6141 6234 1 OUTPUTS
6142 6235 1 A pointer to an RST entry for the RSTPTR object is returned as the
6143 6236 1 routine value. This RST entry will have the proper invocation
6144 6237 1 number for the object. The returned pointer is identical to
6145 6238 1 RSTPTR if the invocation number is zero. RSTPTR is also re-
6146 6239 1 turned unchanged if it does not point to a data object.
6147 6240 1
6148 6241 1
6149 6242 1 BEGIN
6150 6243 2
6151 6244 2 MAP
6152 6245 2 RSTPTR: REF RST$ENTRY, ! Pointer to RST entry for data object
6153 6246 2 ! whose invocation number is to be
6154 6247 2 ! determined
6155 6248 2 SCOPE_RSTPTR: REF RST$ENTRY; ! Pointer to the RST entry which
6156 6249 2 ! defines the scope in which
6157 6250 2 ! the object is to be found
6158 6251 2
6159 6252 2 OWN
6160 6253 2 SPVALUE: REF VECTOR[,LONG]; ! Current call frame's SP value
6161 6254 2
6162 6255 2 LOCAL
6163 6256 2 CURRENT_REG: REF VECTOR[,LONG], ! Pointer to vector of current register
6164 6257 2 ! values (at top of stack)
6165 6258 2 DSTPTR: REF DST$RECORD, ! Pointer to Static Link DST record
6166 6259 2 FRAME_FOUND_FLAG, ! Set to TRUE when a call frame for a
6167 6260 2 ! desired routine has been found

```

6167	6261	2	FRAMEPTR: REF BLOCK[BYTE],	Pointer to current VAX call frame
6168	6262	2	INVPTR: REF RST\$ENTRY,	Pointer to Invocation Number RST Entry
6169	6263	2	J,	Call frame register-vector index
6170	6264	2	PATHNAME,	Pointer to data item Pathname Descr.
6171	6265	2	PATHSTRING,	Pointer to pathname counted ASCII
6172	6266	2	PCVAL,	Current call frame's PC value
6173	6267	2	REGPTR: REF VECTOR[LONG],	Pointer to a register's save location
6174	6268	2	REGSAVELOC: REF VECTOR[LONG],	Pointer to call frame register save
6175	6269	2		area for registers R0 - R11
6176	6270	2	REGVEC: VECTOR[17, LONG],	Vector of pointers to save areas for
6177	6271	2		the current frame's registers
6178	6272	2	ROUTPTR: REF RST\$ENTRY,	Pointer to RST entry for routine which
6179	6273	2		declares the RSTPTR data item
6180	6274	2	ROUT_INVOC_COUNT,	Invocation count of ROUTPTR routine
6181	6275	2	RPTR: REF RST\$ENTRY,	Pointer to RST entry for possible
6182	6276	2		nested routine
6183	6277	2	RUNFRAME_PTR,	Pointer to current entry in CALL com-
6184	6278	2		mand runframe stack (needed by
6185	6279	2		the GET_REGISTER_VALUES routine)
6186	6280	2	SATPTR: REF SAT\$ENTRY,	Pointer to Static Address Table entry
6187	6281	2	SAVEREGSYMID,	Save area for DBG\$REG_SYMID
6188	6282	2	SAVEREGVAL: VECTOR[17, LONG],	Save area for DBG\$REG_VALUES
6189	6283	2	SAVEREGVEC: VECTOR[17, LONG],	Save area for DBG\$REG_VECTOR
6190	6284	2	SCOPE: REF RST\$ENTRY,	Pointer to scope RST entry
6191	6285	2	SCOPE_INVOC_COUNT,	Invocation count of SCOPE routine
6192	6286	2	SCOPE_INVOC_NUM,	Invocation number we are looking for
6193	6287	2		of routine pointed to by SCOPE
6194	6288	2	STATIC_LINK_FP,	Frame Pointer value from Static Link
6195	6289	2		DST record
6196	6290	2	VALKIND,	Value kind returned by DBG\$STA_VALSPEC
6197	6291	2	VALVECTOR: VECTOR[3, LONG];	Value vector returned by VALSPEC rou-
6198	6292	2		ine: byte address, bit offset,
6199	6293	2		and frame pointer value.
6200	6294	2		
6201	6295	2		
6202	6296	2		
6203	6297	2	! If RSTPTR does not point to a Data Item RST Entry, we return it unchanged	
6204	6298	2	! since invocation numbers are only meaningful for data objects.	
6205	6299	2		
6206	6300	2	IF .RSTPTR[RST\$B_KIND] NEQ RST\$K_DATA THEN RETURN .RSTPTR;	
6207	6301	2		
6208	6302	2		
6209	6303	2	! If the scope is anything other than a routine or a block in a routine,	
6210	6304	2	! it cannot have an associated invocation number. We thus return the input	
6211	6305	2	! RST pointer without change.	
6212	6306	2		
6213	6307	2	IF .SCOPE_RSTPTR EQL 0 THEN RETURN .RSTPTR;	
6214	6308	2	IF (.SCOPE_RSTPTR[RST\$B_KIND] NEQ RST\$K_ROUTINE) AND	
6215	6309	2	(.SCOPE_RSTPTR[RST\$B_KIND] NEQ RST\$K_BLOCK)	
6216	6310	2	THEN	
6217	6311	2	RETURN .RSTPTR;	
6218	6312	2		
6219	6313	2		
6220	6314	2	! Get the invocation number associated with the scope RST entry.	
6221	6315	2		
6222	6316	2	SCOPE = .SCOPE_RSTPTR;	
6223	6317	2	SCOPE_INVOC_NUM = 0;	

```

6224 6318 2 IF .SCOPE[RST$V_INVOCNUM]
6225 6319 2 THEN
6226 6320 2 BEGIN
6227 6321 2 INVPTR = .SCOPE[RST$L_SYMCHNPTR];
6228 6322 2 SCOPE_INVOC_NUM = .INVPTR[RST$L_INVOCNUM];
6229 6323 2 SCOPE = .INVPTR[RST$L_UPSCOPEPTR];
6230 6324 2 END;
6231 6325 2
6232 6326 2
6233 6327 2 ! If SCOPE points to a lexical block, find the nearest up-scope routine.
6234 6328 2 ! This is the routine to which the scope's invocation number applies.
6235 6329 2
6236 6330 2 WHILE .SCOPE[RST$B_KIND] NEQ RST$K_ROUTINE DO
6237 6331 2 BEGIN
6238 6332 2 IF .SCOPE[RST$B_KIND] EQL RST$K_MODULE THEN RETURN .RSTPTR;
6239 6333 2 SCOPE = .SCOPE[RST$L_UPSCOPEPTR];
6240 6334 2 END;
6241 6335 2
6242 6336 2
6243 6337 2 ! Get a pointer to the RST entry for the innermost routine up-scope from
6244 6338 2 ! the data object RST entry. This is the routine which immediately contains
6245 6339 2 ! the desired data object.
6246 6340 2
6247 6341 2 ROUTPTR = .RSTPTR;
6248 6342 2 WHILE .ROUTPTR[RST$B_KIND] NEQ RST$K_ROUTINE DO
6249 6343 2 BEGIN
6250 6344 2 IF .ROUTPTR[RST$B_KIND] EQL RST$K_MODULE THEN RETURN .RSTPTR;
6251 6345 2 ROUTPTR = .ROUTPTR[RST$L_UPSCOPEPTR];
6252 6346 2 END;
6253 6347 2
6254 6348 2
6255 6349 2 ! If that innermost routine is the desired scope, we build a new RST entry
6256 6350 2 ! for the data item with the scope's invocation number and return that.
6257 6351 2
6258 6352 2 IF .ROUTPTR EQL .SCOPE
6259 6353 2 THEN
6260 6354 2 BEGIN
6261 6355 2 IF .SCOPE_INVOC_NUM EQL 0 THEN RETURN .RSTPTR;
6262 6356 2 RETURN DBG$BUILD_INVOC_RST(.RSTPTR, .SCOPE_INVOC_NUM);
6263 6357 2 END;
6264 6358 2
6265 6359 2
6266 6360 2 ! The innermost routine and the desired scope are different. We must thus
6267 6361 2 ! go through the VAX call stack to find the proper ROUTPTR frame to go with
6268 6362 2 ! the SCOPE we are starting with. This requires us to follow static links
6269 6363 2 ! where present to do the up-level addressing correctly.
6270 6364 2
6271 6365 2 ! We start by initializing the current stack frame's Program Counter (PC),
6272 6366 2 ! Frame Pointer (FP), and other register values. We also initialize the
6273 6367 2 ! pointer into the CALL command runframe-stack.
6274 6368 2
6275 6369 2 PCVAL = .DBG$RUNFRAME[DBG$USER_PC];
6276 6370 2 FRAMEPTR = .DBG$RUNFRAME[DBG$USER_FP];
6277 6371 2 CURRENT_REG = DBG$RUNFRAME[DBG$USER_REGS];
6278 6372 2 RUNFRAME_PTR = .DBG$RUNFRAME[DBG$NEXT_LINK];
6279 6373 2 INCR I FROM 0 TO 16 DO REGVEC[I] = CURRENT_REG[I];
6280 6374 2
```

6281	6375	2
6282	6376	2
6283	6377	2
6284	6378	2
6285	6379	2
6286	6380	2
6287	6381	2
6288	6382	2
6289	6383	2
6290	6384	3
6291	6385	3
6292	6386	3
6293	6387	3
6294	6388	3
6295	6389	3
6296	6390	4
6297	6391	3
6298	6392	4
6299	6393	4
6300	6394	4
6301	6395	4
6302	6396	3
6303	6397	3
6304	6398	3
6305	6399	3
6306	6400	3
6307	6401	3
6308	6402	3
6309	6403	3
6310	6404	3
6311	6405	4
6312	6406	3
6313	6407	4
6314	6408	4
6315	6409	4
6316	6410	4
6317	6411	4
6318	6412	4
6319	6413	4
6320	6414	4
6321	6415	4
6322	6416	4
6323	6417	4
6324	6418	4
6325	6419	4
6326	6420	4
6327	6421	4
6328	6422	4
6329	6423	4
6330	6424	4
6331	6425	4
6332	6426	4
6333	6427	4
6334	6428	4
6335	6429	4
6336	6430	4
6337	6431	4

Search through the VAX call stack looking for the SCOPE routine's call frame and then the ROUTPTR call frame up-scope from it. Pick up all register save area addresses in the stack along the way.

```
ROUT_INVOC_COUNT = 0;
SCOPE_INVOC_COUNT = 0;
STATIC_LINK_FP = 0;
WHILE TRUE DO
  BEGIN
```

If we got to the bottom of the stack without finding the desired invocation of the ROUTPTR routine, report an error.

```
IF (.PCVAL EQL 0) OR (.FRAMEPTR[SFS$HANDLER] EQL DBG$FINAL_HANDL)
  THEN
```

```
  BEGIN
    DBG$STA SYMPATHNAME(.RSTPTR, PATHNAME);
    DBG$NPATHDESC TO CS(.PATHNAME, PATHSTRING);
    SIGNAL(DBG$PROFRNOT, 1, .PATHSTRING);
  END;
```

Check to see if the current call frame is a frame for the routine currently pointed to by SCOPE. If so, find the static link (if any) and make SCOPE point to the RST entry of the routine immediately up-scope from the routine currently pointed to by SCOPE.

```
IF (.PCVAL GEQU .SCOPE[RST$STARTADDR]) AND
    (.PCVAL LEQU .SCOPE[RST$ENDADDR])
  THEN
    BEGIN
```

The current PC value is in the range of the SCOPE routine, so we set FRAME_FOUND_FLAG. However, this frame could actually be for a nested routine within the SCOPE routine. We check for that and clear FRAME_FOUND_FLAG if that turns out to be the case.

```
FRAME_FOUND_FLAG = TRUE;
SATPTR = .SCOPE[RST$RTNSATPTR];
```

WARNING -- We can get into trouble here. Previously, we have assumed that the SAT is always around. This may not be the case if this module has been canceled. There are times when the module could be canceled and then set again to make us believe the the SAT is valid for this RST, but it is not! To correct the problem, when a module is canceled the field RST\$RTNSATPTR is set to ZERO for each routine. So if the module for this RST has been canceled, SATPTR will be zero from the above statement. The problem is that this assumes there are no nested routines that truly require the correct context information. This is, of course, WRONG. A way of saving and getting to the SAT information must be found in the future. B.A. Becker MAY-1984

```
IF .SATPTR NEQ 0
THEN
    SATPTR = .SATPTR[SAT$L_FLINK];

WHILE TRUE DO
    BEGIN
        IF .SATPTR EQL 0 THEN EXITLOOP;
        IF (.PCVAL LSSU .SATPTR[SAT$L_START]) THEN EXITLOOP;
        RPTR = .SATPTR[SAT$L_RSTPTR];
        IF (.PCVAL LEQU .SATPTR[SAT$L_END]) AND
            (.RPTR[RST$B_KIND] EQL RST$K_ROUTINE)
        THEN
            BEGIN
                FRAME_FOUND_FLAG = FALSE;
                EXITLOOP;
            END;

        SATPTR = .SATPTR[SAT$L_FLINK];
    END;

! If this call frame really is for the SCOPE routine, we see if it
! is the invocation we are looking for.
IF .FRAME_FOUND_FLAG
THEN
    BEGIN

        ! If this is the invocation we are looking for, determine which
        ! invocation of the next routine up-scope from SCOPE to look
        ! for next.
        IF (.STATIC_LINK_FP EQL .FRAMEPTR) OR
            (.SCOPE_INVOC_COUNT EQL .SCOPE_INVOC_NUM)
        THEN
            BEGIN

                ! This is the invocation of the SCOPE routine we want. If
                ! this frame is also a frame for the ROUTPTR routine, we
                ! have found the call frame we want for the data item. We
                ! thus exit the loop searching through the call stack.
                IF .SCOPE EQL .ROUTPTR THEN EXITLOOP;

                ! If no Static Link DST record was specified, we want to
                ! look for the first invocation in the stack of the routine
                ! up-scope from the SCOPE routine. We set SCOPE_INVOC_NUM
                ! and SCOPE_INVOC_COUNT to make this happen.
                IF .SCOPE[RST$L_STATIC_LINK] EQL 0
                THEN
                    BEGIN
                        SCOPE_INVOC_NUM = 1;
                        SCOPE_INVOC_COUNT = 0;
```

```
6338 6432 4
6339 6433 4
6340 6434 4
6341 6435 4
6342 6436 4
6343 6437 5
6344 6438 5
6345 6439 5
6346 6440 5
6347 6441 5
6348 6442 6
6349 6443 5
6350 6444 6
6351 6445 6
6352 6446 6
6353 6447 5
6354 6448 5
6355 6449 5
6356 6450 4
6357 6451 4
6358 6452 4
6359 6453 4
6360 6454 4
6361 6455 4
6362 6456 4
6363 6457 4
6364 6458 5
6365 6459 5
6366 6460 5
6367 6461 5
6368 6462 5
6369 6463 5
6370 6464 5
6371 6465 5
6372 6466 6
6373 6467 5
6374 6468 6
6375 6469 6
6376 6470 6
6377 6471 6
6378 6472 6
6379 6473 6
6380 6474 6
6381 6475 6
6382 6476 6
6383 6477 6
6384 6478 6
6385 6479 6
6386 6480 6
6387 6481 6
6388 6482 6
6389 6483 6
6390 6484 6
6391 6485 6
6392 6486 7
6393 6487 7
6394 6488 7
```

```

: 6395      6489 7      STATIC_LINK_FP = 0;
: 6396      6490 7      END
: 6397      6491 7
: 6398      6492 7
: 6399      6493 7
: 6400      6494 7
: 6401      6495 7
: 6402      6496 7
: 6403      6497 7
: 6404      6498 6
: 6405      6499 7
: 6406      6500 7
: 6407      6501 7
: 6408      6502 7
: 6409      6503 7
: 6410      6504 7
: 6411      6505 7
: 6412      6506 7
: 6413      6507 7
: 6414      6508 7
: 6415      6509 7
: 6416      6510 7
: 6417      6511 8
: 6418      6512 8
: 6419      6513 8
: 6420      6514 8
: 6421      6515 8
: 6422      6516 8
: 6423      6517 7
: 6424      6518 7
: 6425      6519 7
: 6426      6520 7
: 6427      6521 7
: 6428      6522 7
: 6429      6523 7
: 6430      6524 7
: 6431      6525 7
: 6432      6526 7
: 6433      6527 7
: 6434      6528 7
: 6435      6529 7
: 6436      6530 7
: 6437      6531 7
: 6438      6532 7
: 6439      6533 8
: 6440      6534 8
: 6441      6535 8
: 6442      6536 7
: 6443      6537 7
: 6444      6538 6
: 6445      6539 6
: 6446      6540 6
: 6447      6541 6
: 6448      6542 6
: 6449      6543 6
: 6450      6544 6
: 6451      6545 6

      STATIC_LINK_FP = 0;
      END

      ! But if a Static Link DST record was specified for this
      ! routine, we use the Value Spec in that DST record to pick
      ! up the Static Link (in the form of a Frame Pointer). We
      ! disable looking for the first up-scope invocation.
      ELSE
      BEGIN
      SCOPE_INVOC_NUM = 0;
      SCOPE_INVOC_COUNT = 0;

      ! Save the current register values and pointers set up
      ! by DBG$STA_SETCONTEXT. Then substitute our own regis-
      ! ter set in the arrays used in Value Spec evaluation.
      SAVEREGSYMID = .DBG$REG_SYMID;
      DBG$REG_SYMID = .RSTPTR;
      INCR I FROM 0 TO 16 DO
      BEGIN
      SAVEREGVEC[I] = .DBG$REG_VECTOR[I];
      SAVEREGVAL[I] = .DBG$REG_VALUES[I];
      DBG$REG_VECTOR[I] = .REGVEC[I];
      REGPTR = .REGVEC[I];
      IF .REGPTR NEQ 0 THEN DBG$REG_VALUES[I] = .REGPTR[0];
      END;

      ! Evaluate the Static Link Value Spec. This produces a
      ! pointer to the desired up-scope call frame.
      DSTPTR = .SCOPE[RST$SL_STATIC_LINK];
      DBG$STA_VALSPEC(DSTPTR[DST$SL_VALSPEC],
      VALVECTOR, VALKIND);
      STATIC_LINK_FP = .VALVECTOR[0];

      ! Restore the saved register values and pointers.
      DBG$REG_SYMID = .SAVEREGSYMID;
      INCR I FROM 0 TO 16 DO
      BEGIN
      DBG$REG_VECTOR[I] = .SAVEREGVEC[I];
      DBG$REG_VALUES[I] = .SAVEREGVAL[I];
      END;

      END;      ! End of Static Link evaluation

      ! Follow the up-scope pointer from the SCOPE routine's RST
      ! entry to the next routine up-scope. Set SCOPE to point
      ! to that routine's Routine RST Entry.
      SCOPE = .SCOPE[RST$SL_UPSCOPEPTR];
```



```
6452 6546 6      WHILE .SCOPE[RST$B_KIND] NEQ RST$K_ROUTINE DO
6453 6547 7      BEGIN
6454 6548 7      IF .SCOPE[RST$B_KIND] EQL RST$K_MODULE
6455 6549 7      THEN
6456 6550 7          $DBG_ERROR('RSTACCESS\FOLLOW_STATIC_LINK');
6457 6551 7
6458 6552 7      SCOPE = .SCOPE[RST$L_UPSCOPEPTR];
6459 6553 6      END;
6460 6554 6
6461 6555 5      END;                                ! End of STATIC_LINK_FP IF-statement
6462 6556 5
6463 6557 5      ! We now know what routine and frame to look for next. Incre-
6464 6558 5      ! ment the invocation count for the current SCOPE routine.
6465 6559 5      SCOPE_INVOC_COUNT = .SCOPE_INVOC_COUNT + 1;
6466 6560 5
6467 6561 5      END;                                ! End of FRAME_FOUND_FLAG IF-statement
6468 6562 5
6469 6563 4      END;                                ! End of PCVAL in SCOPE IF-statement
6470 6564 4
6471 6565 3      END;
6472 6566 3
6473 6567 3      ! Check to see if the current call frame is a frame for the ROUTPTR
6474 6568 3      ! routine (but not for a nested routine within the ROUTPTR routine).
6475 6569 3      ! If so, increment the ROUTPTR routine's invocation count. This code
6476 6570 3      ! thus computes the data item's final invocation count.
6477 6571 3
6478 6572 3      IF (.PCVAL GEQU .ROUTPTR[RST$L_STARTADDR]) AND
6479 6573 3      (.PCVAL LEQU .ROUTPTR[RST$L_ENDADDR])
6480 6574 4      THEN
6481 6575 3      BEGIN
6482 6576 4      FRAME_FOUND_FLAG = TRUE;
6483 6577 4      SATPTR = .ROUTPTR[RST$L_RTNSATPTR];
6484 6578 4
6485 6579 4      ! WARNING -- We can get into trouble here. Previously, we have
6486 6580 4      ! assumed that the SAT is always around. This may not be the
6487 6581 4      ! case if this module has been canceled. There are times when
6488 6582 4      ! the module could be canceled and then set again to make us
6489 6583 4      ! believe the the SAT is valid for this RST, but it is not! To
6490 6584 4      ! correct the problem, when a module is canceled the field
6491 6585 4      ! RST$L_RTNSATPTR is set to ZERO for each routine.
6492 6586 4      ! So if the module for this RST has been canceled, SATPTR will
6493 6587 4      ! be zero from the above statement. The problem is that this
6494 6588 4      ! assumes there are no nested routines that truly require the
6495 6589 4      ! correct context information. This is, of course, WRONG. A
6496 6590 4      ! way of saving and getting to the SAT information must be
6497 6591 4      ! found in the future. B.A. Becker MAY-1984
6498 6592 4
6499 6593 4      IF .SATPTR NEQ 0
6500 6594 4      THEN
6501 6595 4      SATPTR = .SATPTR[SAT$L_FLINK];
6502 6596 4
6503 6597 4      WHILE TRUE DO
6504 6598 4      BEGIN
6505 6599 5      IF .SATPTR EQL 0 THEN EXITLOOP;
6506 6600 5      IF (.PCVAL LSSU .SATPTR[SAT$L_START]) THEN EXITLOOP;
6507 6601 5      RPTR = .SATPTR[SAT$L_RSTPTR];
6508 6602 5
```

```
6509      6603 5      IF (.PCVAL LEQU .SATPTR[SAT$L_END]) AND
6510      6604 6      (.RPTR[RST$B_KIND] EQL RST$K_ROUTINE)
6511      6605 5      THEN
6512      6606 6          BEGIN
6513      6607 6              FRAME_FOUND_FLAG = FALSE;
6514      6608 6              EXITLOOP;
6515      6609 5              END;
6516      6610 5
6517      6611 5          SATPTR = .SATPTR[SAT$L_FLINK];
6518      6612 4          END;
6519      6613 4
6520      6614 4      IF .FRAME_FOUND_FLAG
6521      6615 4      THEN
6522      6616 4          RCUT_INVOC_COUNT = .ROUT_INVOC_COUNT + 1;
6523      6617 4
6524      6618 3      END;
6525      6619 3
6526      6620 3
6527      6621 3      ! Pick up the addresses of the register save areas in this call frame.
6528      6622 3      ! Save those addresses in REGVEC. This allows us to keep track of the
6529      6623 3      ! current register values as we go on to the next frame in the stack.
6530      6624 3
6531      6625 3      GET_REGISTER_VALUES(.FRAMEPTR, RUNFRAME_PTR, REGVEC);
6532      6626 3
6533      6627 3
6534      6628 3      ! Determine what the value of SP (the Stack Pointer) is for the current
6535      6629 3      ! CALL frame and save that in the OWN variable SPVALUE. Then make the
6536      6630 3      ! save-location pointer in REGVEC point to SPVALUE. (Since SP does not
6537      6631 3      ! have a true save-location, the OWN variable fakes one.)
6538      6632 3
6539      6633 3      REGPTR = .REGVEC[14];
6540      6634 3      SPVALUE = .REGPTR[0];
6541      6635 3      REGVEC[14] = SPVALUE;
6542      6636 3
6543      6637 3
6544      6638 3      ! Dig out the values of PC and FP for the current CALL frame. Then
6545      6639 3      ! loop for the next stack frame.
6546      6640 3
6547      6641 3      REGPTR = .REGVEC[15];
6548      6642 3      PCVAL = .REGPTR[0];
6549      6643 3      REGPTR = .REGVEC[13];
6550      6644 3      FRAMEPTR = .REGPTR[0];
6551      6645 3
6552      6646 2      END;
6553      6647 2
6554      6648 2
6555      6649 2      ! We have now found the proper invocation number for the ROUTPTR routine.
6556      6650 2      ! We thus build an RST entry for the data item with that invocation number
6557      6651 2      ! (if necessary) and return a pointer to that RST entry.
6558      6652 2
6559      6653 2      IF .ROUT_INVOC_COUNT EQL 0 THEN RETURN .RSTPTR;
6560      6654 2      RETURN DBG$BUILT_INVOC_RST(.RSTPTR, .ROUT_INVOC_COUNT);
6561      6655 2
6562      6656 1      END;
```

4C 4C 4F 46 5C 53 53 45 43 43 41 54 53 52 1C 00424 P.AED: .PSECT DBG\$PLIT,NOWRT, SHR, PIC,0
4B 4E 49 4C 5F 43 49 54 41 54 53 5F 57 4F 00433 .ASCII <28>\RSTACCESS\<92>\FOLLOW_STATIC_LINK\ ;

.PSECT DBG\$OWN,NOEXE, PIC,2
00054 SPVALUE:.BLKB 4

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

OFFC 00000 FOLLOW_STATIC_LINK:
5E FF00 CE 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 : 6204
58 04 AC D0 00007 MOVAB -256(SP), SP : 6300
06 14 A8 91 0000B MOVL RSTPTR, R8 :
10 12 0000F CMPB 20(R8), #6 :
50 08 AC D0 00011 BNEQ 1\$: 6307
02 14 A0 91 00017 MOVL SCOPE_RSTPTR, R0 :
09 13 00015 BEQL 8\$: 6308
03 14 A0 91 0001B CMPB 20(R0), #2 :
09 13 0001B BEQL 3\$: 6309
03 13 0001D CMPB 20(R0), #3 :
03 13 00021 BEQL 3\$:
022B 31 00023 BRW 31\$:
53 50 D0 00026 MOVL R0, SCOPE : 6316
08 AE D4 00029 CLRL SCOPE_INVOC_NUM : 6317
15 A3 02 E1 0002C BBC #2, 2T(SCOPE), 4\$: 6318
50 08 A3 D0 00031 MOVL 8(SCOPE), INVPTR : 6321
08 AE 18 A0 D0 00035 MOVL 24(INVPTR), SCOPE_INVOC_NUM : 6322
53 10 A0 D0 0003A MOVL 16(INVPTR), SCOPE : 6323
02 14 A3 91 0003E CMPB 20(SCOPE), #2 : 6330
0C 13 00042 BEQL 5\$:
01 14 A3 91 00044 CMPB 20(SCOPE), #1 : 6332
D9 13 00048 BEQL 2\$:
53 10 A3 D0 0004A MOVL 16(SCOPE), SCOPE : 6333
EE 11 0004E BRB 4\$: 6330
54 58 D0 00050 MOVL R8, ROUTPTR : 6341
02 14 A4 91 00053 CMPB 20(ROUTPTR), #2 : 6342
0C 13 00057 BEQL 7\$:
01 14 A4 91 00059 CMPB 20(ROUTPTR), #1 : 6344
C4 13 0005D BEQL 2\$:
54 10 A4 D0 0005F MOVL 16(ROUTPTR), ROUTPTR : 6345
EE 11 00063 BRB 6\$: 6342
53 54 D1 00065 CMPL ROUTPTR, SCOPE : 6352
0B 12 00068 BNEQ 9\$:
08 AE D5 0006A TSTL SCOPE_INVOC_NUM : 6355
B4 13 0006D BEQL 2\$:
08 AE DD 0006F PUSHL SCOPE_INVOC_NUM : 6356
01E3 31 00072 BRW 33\$:
5A 00000000G 00 D0 00075 MOVL DBG\$RUNFRAME+64, PCVAL : 6369
0C AE 00000000G 00 D0 0007C MOVL DBG\$RUNFRAME+56, FRAMEPTR : 6370
51 00000000G 00 9E 00084 MOVAB DBG\$RUNFRAME+4, CURRENT_REG : 6371
24 AE 00000000G 00 D0 0008B MOVL DBG\$RUNFRAME, RUNFRAME_PTR : 6372
50 D4 00093 CLRL I : 6373
BC AD40 6140 DE 00095 MOVAL (CURRENT_REG)[1], REGVEC[1] :

F6	50	10	F3	0009B	AOBLEQ	#16, 1, 10\$	
		14	AE	D4	0009F	CLRL	ROUT_INVOC_COUNT
			6E	7C	0C0A2	CLRQ	STATIC_LINK_FP
			5A	D5	000A4	11\$: TSTL	PCVAL
			0D	13	000A6	BEQL	12\$
	50	00000000G	00	9E	000A8	MOVAB	DBG\$FINAL_HANDL, R0
	50	0C	BE	D1	000AF	CMPL	@FRAMEPTR, R0
			29	12	000B3	BNEQ	13\$
		18	AE	9F	000B5	12\$: PUSHAB	PATHNAME
			58	DD	000B8	PUSHL	R8
F1C6	CF		02	FB	000BA	CALLS	#2, DBG\$STA_SYMPATHNAME
		1C	AE	9F	000BF	PUSHAB	PATHSTRING
		1C	AE	DD	000C2	PUSHL	PATHNAME
00000000G	00		02	FB	000C5	CALLS	#2, DBG\$NPATHTDESC_TO_CS
		1C	AE	DD	000CC	PUSHL	PATHSTRING
			01	DD	000CF	PUSHL	#1
00000000G	00	00028CB0	8F	DD	000D1	PUSHL	#167088
	18	A3	03	FB	00CD7	CALLS	#3, LIB\$SIGNAL
			5A	D1	000DE	13\$: CMPL	PCVAL, 24(SCOPE)
			03	1E	000E2	BGEQU	15\$
		00FA	31	000E4	14\$: BRW	26\$	
	1C	A3	5A	D1	000E7	15\$: CMPL	PCVAL, 28(SCOPE)
			F7	1A	000EB	BGTRU	14\$
	59		01	D0	000ED	MOVL	#1, FRAME_FOUND_FLAG
	52	20	A3	D0	000F0	MOVL	32(SCOPE), SATPTR
			1D	13	000F4	BEQL	17\$
	52		62	D0	000F6	16\$: MOVL	(SATPTR), SATPTR
			18	13	000F9	BEQL	17\$
04	A2		5A	D1	000FB	CMPL	PCVAL, 4(SATPTR)
			12	1F	000FF	BLSSU	17\$
	55	0C	A2	D0	00101	MOVL	12(SATPTR), RPTR
08	A2		5A	D1	00105	CMPL	PCVAL, 8(SATPTR)
			EB	1A	00109	BGTRU	16\$
	02	14	A5	91	0010B	CMPB	20(RPTR), #2
			E5	12	0010F	BNEQ	16\$
			59	D4	00111	CLRL	FRAME_FOUND_FLAG
	CE		59	E9	00113	17\$: BLBC	FRAME_FOUND_FLAG, 14\$
0C	AE		6E	D1	00116	CMPL	STATIC_LINK_FP, FRAMEPTR
			0A	13	0011A	BEQL	18\$
08	AE	04	AE	D1	0011C	CMPL	SCOPE_INVOC_COUNT, SCOPE_INVOC_NUM
			03	13	00121	BEQL	18\$
		00B8	31	00123	BRW	25\$	
	54		53	D1	00126	18\$: CMPL	SCOPE, ROUTPTR
			03	12	00129	BNEQ	19\$
		011E	31	0012B	BRW	30\$	
		04	AE	D4	0012E	19\$: CLRL	SCOPE_INVOC_COUNT
		24	A3	D5	00131	TSTL	36(SCOPE)
			08	12	00134	BNEQ	20\$
08	AE		01	D0	00136	MOVL	#1, SCOPE_INVOC_NUM
			6E	D4	0013A	CLRL	STATIC_LINK_FP
			79	11	0013C	BRB	24\$
		08	AE	D4	0013E	20\$: CLRL	SCOPE_INVOC_NUM
10	AE	00000000	EF	D0	00141	MOVL	DBG\$REG_SYMTD, SAVEREGSYMID
00000000	EF		58	D0	00149	MOVL	R8, DBG\$REG_SYMTD
			50	D4	00150	CLRL	I
	56	00000000G	0040	DE	00152	21\$: MOVAL	DBG\$REG_VECTOR[I], R6
34	AE40		66	D0	0015A	MOVL	(R6), SAVEREGVEC[I]

	51	00000000G0040	DE	0015F	MOVAL	DBG\$REG_VALUES[I], R1	: 6513	
78	AE40		61	D0	00167	MOVL	(R1), SAVEREGVAL[I]	: 6514
	66	BC AD40	D0	0016C	MOVL	REGVEC[I], (R6)	: 6515	
	57	BC AD40	D0	00171	MOVL	REGVEC[I], REGPTR	: 6516	
			03	13	00176	BEQL	22\$: 6510
	61		67	D0	00178	MOVL	(REGPTR), (R1)	: 6523
D3	50		10	F3	0017B	AOBLEQ	#16, I, 21\$: 6524
	5B	24	A3	D0	0017F	MOVL	36(SCOPE), DSTPTR	: 6526
		20	AE	9F	00183	PUSHAB	VALKIND	: 6531
		2C	AE	9F	00186	PUSHAB	VALVECTOR	: 6532
		02	AB	9F	00189	PUSHAB	2(DSTPTR)	: 6534
F5FD	CF		03	FB	0018C	CALLS	#3, DBG\$STA_VALSPEC	: 6535
	6E	28	AE	D0	00191	MOVL	VALVECTOR, STATIC_LINK_FP	: 6532
00000000'	EF	10	AE	D0	00195	MOVL	SAVEREGSYMID, DBG\$REG_SYMID	: 6534
			50	D4	0019D	CLRL	I	: 6535
00000000G0040		34	AE40	D0	0019F	MOVL	SAVEREGVEC[I], DBG\$REG_VECTOR[I]	: 6532
00000000G0040		78	AE40	D0	001A9	MOVL	SAVEREGVAL[I], DBG\$REG_VALUES[I]	: 6545
E8			10	F3	001B3	AOBLEQ	#16, I, 23\$: 6546
	53	10	A3	D0	001B7	MOVL	16(SCOPE), SCOPE	: 6548
	02	14	A3	91	001BB	CMPB	20(SCOPE), #2	: 6550
			1D	13	001BF	BEQL	25\$: 6552
	01	14	A3	91	001C1	CMPB	20(SCOPE), #1	: 6561
			F0	12	001C5	BNEQ	24\$: 6573
	00000000'		EF	9F	001C7	PUSHAB	P.AED	: 6574
			01	DD	001CD	PUSHL	#1	: 6577
	00028362		8F	DD	001CF	PUSHL	#164706	: 6578
00000000G	00		03	FB	001D5	CALLS	#3, LIB\$SIGNAL	: 6594
			D9	11	001DC	BRB	24\$: 6596
		04	AE	D6	001DE	INCL	SCOPE_INVOC_COUNT	: 6600
18	A4		5A	D1	001E1	CMPL	PCVAL, 24(ROUTPTR)	: 6601
			32	1F	001E5	BLSSU	29\$: 6602
1C	A4		5A	D1	001E7	CMPL	PCVAL, 28(ROUTPTR)	: 6603
			2C	1A	001EB	BGTRU	29\$: 6604
	59		01	D0	001ED	MOVL	#1, FRAME_FOUND_FLAG	: 6607
	52	20	A4	D0	001F0	MOVL	32(ROUTPTR), SATPTR	: 6614
			1D	13	001F4	BEQL	28\$: 6616
	52		62	D0	001F6	MOVL	(SATPTR), SATPTR	: 6625
			18	13	001F9	BEQL	28\$: 6633
04	A2		5A	D1	001FB	CMPL	PCVAL, 4(SATPTR)	: 6634
			12	1F	001FF	BLSSU	28\$: 6635
	55	0C	A2	D0	00201	MOVL	12(SATPTR), RPTR	: 6641
08	A2		5A	D1	00205	CMPL	PCVAL, 8(SATPTR)	: 6642
			EB	1A	00209	BGTRU	27\$: 6642
	02	14	A5	91	0020B	CMPB	20(RPTR), #2	: 6642
			E5	12	0020F	BNEQ	27\$: 6642
			59	D4	00211	CLRL	FRAME_FOUND_FLAG	: 6642
	03		59	E9	00213	BLBC	FRAME_FOUND_FLAG, 29\$: 6642
		14	AE	D6	00216	INCL	ROUT_INVOC_COUNT	: 6642
		BC	AD	9F	00219	PUSHAB	REGVEC	: 6642
		28	AE	9F	0021C	PUSHAB	RUNFRAME_PTR	: 6642
		14	AE	DD	0021F	PUSHL	FRAMEPTR	: 6642
0000V	CF		03	FB	00222	CALLS	#3, GET_REGISTER_VALUES	: 6642
	57	F4	AD	D0	00227	MOVL	REGVEC+56, REGPTR	: 6642
00000000'	EF		67	D0	0022B	MOVL	(REGPTR), SPVALUE	: 6642
F4	AD	00000000'	EF	9E	00232	MOVAB	SPVALUE, REGVEC+56	: 6642
	57	F8	AD	D0	0023A	MOVL	REGVEC+60, REGPTR	: 6642
	5A		67	D0	0023E	MOVL	(REGPTR), PCVAL	: 6642

RSTACCESS
V04-000

J 1
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 208
(40)

OC	57	F0	AD	D0	00241	MOVL	REGVEC+52, REGPTR	:	6643	
	AE		67	D0	00245	MOVL	(REGPTR), FRAMEPTR	:	6644	
			FE	31	00249	BRW	11\$:	6383	
		14	AE	D5	0024C	30\$:	TSTL	ROUT_INVOC_COUNT	:	6653
			04	12	0024F		BNEQ	32\$:	
	50		58	D0	00251	31\$:	MOVL	R8, R0	:	
			04	00254			RET		:	
		14	AE	DD	00255	32\$:	PUSHL	ROUT_INVOC_COUNT	:	6654
			58	DD	00258	33\$:	PUSHL	R8	:	
D470	CF		02	FB	0025A		CALLS	#2, DBG\$BUILD_INVOC_RST	:	
			04	0025F			RET		:	6656

; Routine Size: 608 bytes, Routine Base: DBG\$CODE + 2A0A

```
6564 6657 1 ROUTINE GET_REGISTER_SYMID(PATHDESCR, SCOPEPTR, REG_LINE_LEX_PTR) =
6565 6658 1
6566 6659 1 FUNCTION
6567 6660 1     This routine determines whether a given input symbol is a register
6568 6661 1     name or not, and if so returns that register's SYMID. The input
6569 6662 1     symbol is identified by a Pathname Descriptor from which the symbol
6570 6663 1     name is extracted. If the symbol name is a register name (R0 - R11,
6571 6664 1     AP, FP, SP, PC, or PSL) with or without a leading percent-sign, a
6572 6665 1     Data RST Entry and a DST record are created for the register. The
6573 6666 1     data type is set to longword integer (DSC$K_DTYPE_L) in the DST record
6574 6667 1     and the name is set to the register name with a leading percent-sign
6575 6668 1     ('%R5' or '%SP', for example).
6576 6669 1
6577 6670 1     The RST entry's up-scope pointer is set as follows. If the register is
6578 6671 1     located in a normal named scope (MCD\ROUT\%R5, for example), the regis-
6579 6672 1     ter RST entry's up-scope pointer is set to point to the RST entry for
6580 6673 1     that scope. If the scope is a numeric scope (such as 0\%R5) which can
6581 6674 1     be converted to a named scope, the same thing is done. However, if the
6582 6675 1     numeric scope cannot be converted to a named scope, meaning that no SET
6583 6676 1     module contains that scope, a dummy 'numeric scope' Module RST Entry is
6584 6677 1     built to represent that scope. This RST entry has the RST$V_MODNUMSCP
6585 6678 1     bit set which is later recognized by DBG$STA_SETCONTEXT as representing
6586 6679 1     an unnamed numeric scope. The name of this 'module' is set to be the
6587 6680 1     scope number in ASCII. Finally, if the scope is the global scope (the
6588 6681 1     GST) or a named scope in a module which is not SET (or does not exist),
6589 6682 1     then the register RST entry is discarded and a zero SYMID is returned
6590 6683 1     (no such symbol).
6591 6684 1
6592 6685 1     The register RST entry and the numeric scope Module RST Entry (if any)
6593 6686 1     are both put on the Temporary RST Entry List. The address of the regis-
6594 6687 1     ter RST entry is then returned as the register's SYMID. If the input
6595 6688 1     symbol does not name a register or does not name a register in a SET
6596 6689 1     module, a zero is returned to indicate this.
6597 6690 1
6598 6691 1 INPUTS
6599 6692 1     PATHDESCR - A pointer to the Pathname Descriptor for the input symbol.
6600 6693 1     This descriptor thus identifies the symbol to be checked for
6601 6694 1     being a register name.
6602 6695 1
6603 6696 1     SCOPEPTR - A pointer to a Scope List Entry for the scope in which the
6604 6697 1     register is located. If there is no such scope, SCOPEPTR
6605 6698 1     is zero.
6606 6699 1
6607 6700 1     REG_LINE_LEX_PTR - If there is a line number in the symbol pathname,
6608 6701 1     this parameter gives the SYMID of the lexical entity named
6609 6702 1     by that line number. If there is no line number, this value
6610 6703 1     is zero.
6611 6704 1
6612 6705 1 OUTPUTS
6613 6706 1     If the input symbol is a register name, an RST Entry is created for
6614 6707 1     this register and its address is returned as the register's
6615 6708 1     SYMID. If the symbol is not a register, zero is returned.
6616 6709 1
6617 6710 1
6618 6711 2 BEGIN
6619 6712 2
6620 6713 2 MAP
```

```

: 6621      6714 2      PATHDESCR: REF PTH$PATHNAME,      : Pointer to symbol Pathname Descriptor
: 6622      6715 2      SCOPEPTR: REF SCOPE$ENTRY;      : Pointer to Scope List Entry
: 6623      6716 2
: 6624      6717 2
: 6625      6718 2      BIND
: 6626      6719 2      REGTBL = UPLIT ('R0 ', 'R1 ', 'R2 ', 'R3 ', : Register name table
: 6627      6720 2      'R4 ', 'R5 ', 'R6 ', 'R7 ', :
: 6628      6721 2      'R8 ', 'R9 ', 'R10 ', 'R11 ', :
: 6629      6722 2      'AP ', 'FP ', 'SP ', 'PC ', :
: 6630      6723 2      'PSL ', 'R12 ', 'R13 ', 'R14 ', :
: 6631      6724 2      'R15 '): VECTOR[,LONG];
: 6632      6725 2
: 6633      6726 2      LOCAL
: 6634      6727 2      DSTNAME: REF VECTOR[,BYTE],      : Pointer to DST record name field
: 6635      6728 2      DSTPTR: REF DST$RECORD,      : Pointer to created register DST record
: 6636      6729 2      LENGTH,      : The byte length of the symbol name
: 6637      6730 2      MODPTR: REF RST$ENTRY,      : Pointer to Module RST Entry or to
: 6638      6731 2      : "numeric scope" Module RST Entry
: 6639      6732 2      NAMEPTR: REF VECTOR[,BYTE],      : Pointer to the symbol's name string
: 6640      6733 2      NEWVALUE,      : Temporary in decimal conversion
: 6641      6734 2      NUMNAME: VECTOR[12, BYTE],      : Numeric scope name in Counted ASCII
: 6642      6735 2      NUMTEMP: VECTOR[12, BYTE],      : Temporary used to compute NUMNAME
: 6643      6736 2      NUMSCP_INVOC_NUM,      : The invocation number associated with
: 6644      6737 2      : a named numeric scope
: 6645      6738 2      PATHSTRING,      : Pointer to pathname Counted ASCII
: 6646      6739 2      PATHVEC: REF VECTOR[,LONG],      : Pointer to pathname vector
: 6647      6740 2      PNAME: REF VECTOR[,BYTE],      : Pointer to Counted ASCII string
: 6648      6741 2      REGNUM,      : The register number of the register
: 6649      6742 2      RNAME: REF VECTOR[,BYTE],      : Pointer to Counted ASCII string
: 6650      6743 2      RPTR: REF RST$ENTRY,      : Temporary RST entry pointer
: 6651      6744 2      RSTPTR: REF RST$ENTRY,      : Pointer to created register RST entry
: 6652      6745 2      SCOPE: REF RST$ENTRY,      : Pointer to RST entry for named scope
: 6653      6746 2      : corresponding to a numeric scope
: 6654      6747 2      SCOPENTRY: SCOPE$ENTRY,      : Local copy of input Scope List Entry
: 6655      6748 2      TEMPNAME: VECTOR[4, BYTE],      : Upcased copy of symbol name
: 6656      6749 2      VALUE;      : The numeric scope number--used to con-
: 6657      6750 2      : vert that number to decimal ASCII
: 6658      6751 2
: 6659      6752 2
: 6660      6753 2      : Check that this Pathname Descriptor describes a symbol name without data
: 6661      6754 2      : qualification and that the symbol contains at least two characters (such
: 6662      6755 2      : as 'R5') and at most four characters (such as '%R11'). If not, this is
: 6663      6756 2      : not a register reference and we return zero.
: 6664      6757 2
: 6665      6758 2      IF .PATHDESCR[PTH$B_PATHCNT] NEQ .PATHDESCR[PTH$B_TOTCNT] THEN RETURN 0;
: 6666      6759 2      PATHVEC = PATHDESCR[PTH$A_PATHVECTOR];
: 6667      6760 2      NAMEPTR = .PATHVEC[.PATHDESCR[PTH$B_PATHCNT] - 1];
: 6668      6761 2      LENGTH = .NAMEPTR[0];
: 6669      6762 2      IF .LENGTH LSS 2 OR .LENGTH GTR 4 THEN RETURN 0;
: 6670      6763 2      NAMEPTR = NAMEPTR[1];
: 6671      6764 2
: 6672      6765 2
: 6673      6766 2      : If the symbol contains a leading percent-sign (as in '%R5'), strip it off.
: 6674      6767 2
: 6675      6768 2      IF .NAMEPTR[0] EQL '%'
: 6676      6769 2      THEN
: 6677      6770 3      BEGIN
```



```

: 6678      6771      3      NAMEPTR = .NAMEPTR + 1;
: 6679      6772      3      LENGTH = .LENGTH - 1;
: 6680      6773      2      END;
: 6681      6774      2
: 6682      6775      2
: 6683      6776      2      ! Copy the symbol name to a temporary buffer. The temporary copy is
: 6684      6777      2      ! upcased so that register names are recognized regardless of case.
: 6685      6778      2      ! This is mainly important for language C.
: 6686      6779      2
: 6687      6780      2      INCR I FROM 0 TO .LENGTH - 1 DO
: 6688      6781      3      BEGIN
: 6689      6782      3      TEMPNAME[I] = .NAMEPTR[I];
: 6690      6783      4      IF (.TEMPNAME[I] GEQ 'a') AND (.TEMPNAME[I] LEQ 'z')
: 6691      6784      3      THEN
: 6692      6785      3      TEMPNAME[I] = .TEMPNAME[I] - 'a' + 'A';
: 6693      6786      3
: 6694      6787      2      END;
: 6695      6788      2
: 6696      6789      2
: 6697      6790      2      ! Loop through the list of valid register names to see if this symbol's
: 6698      6791      2      ! name is a register name.
: 6699      6792      2
: 6700      6793      2      REGNUM = -1;
: 6701      6794      2      INCR I FROM 0 TO 20 DO
: 6702      6795      3      BEGIN
: 6703      6796      3      IF CH$EQL(.LENGTH, TEMPNAME, 3, REGTBL[I], ' ')
: 6704      6797      3      THEN
: 6705      6798      4      BEGIN
: 6706      6799      4      REGNUM = .I;
: 6707      6800      4      EXITLOOP;
: 6708      6801      3      END;
: 6709      6802      3
: 6710      6803      2      END;
: 6711      6804      2
: 6712      6805      2
: 6713      6806      2      ! If the symbol name was not a register name, return zero--not a register.
: 6714      6807      2
: 6715      6808      2      IF .REGNUM LSS 0 THEN RETURN 0;
: 6716      6809      2
: 6717      6810      2
: 6718      6811      2      ! It is a register name. If REGNUM is larger than 16, the user requested
: 6719      6812      2      ! R12, R13, R14, or R15, so we reset REGNUM to point to AP, FP, SP, or PC.
: 6720      6813      2
: 6721      6814      2      IF .REGNUM GTR 16 THEN REGNUM = .REGNUM - 5;
: 6722      6815      2
: 6723      6816      2
: 6724      6817      2      ! Create a Data RST Entry for the register.
: 6725      6818      2
: 6726      6819      2      RSTPTR = DBG$GET_MEMORY(RST$K_DATENTSI2 + (11 + %UPVAL)/%UPVAL);
: 6727      6820      2      DSTPTR = .RSTPTR + RST$K_DATENTSI2*%UPVAL;
: 6728      6821      2      RSTPTR[RST$I_DSTPTR] = .DSTPTR;
: 6729      6822      2      RSTPTR[RST$B_KIND] = RST$K_DATA;
: 6730      6823      2      RSTPTR[RST$V_NONZLENGTH] = TRUE;
: 6731      6824      2      RSTPTR[RST$V_REGISTER] = TRUE;
: 6732      6825      2
: 6733      6826      2
: 6734      6827      2      ! Also create a DST record for the register. This DST record makes
```

```

: 6735      6828 2      ! the data type longword integer and contains the register name
: 6736      6829 2      ! preceded by a percent-sign (e.g., '%R5').
: 6737      6830 2
: 6738      6831 2      DSTPTR[DST$B_LENGTH] = 8 + .LENGTH;
: 6739      6832 2      DSTPTR[DST$B_TYPE] = DSC$K_DTYPE_L;
: 6740      6833 2      DSTPTR[DST$V_VALKIND] = DST$K_VA[KIND_REG];
: 6741      6834 2      DSTPTR[DST$L_VALUE] = .REGNUM;
: 6742      6835 2      DSTPTR[DST$B_NAME] = .LENGTH + 1;
: 6743      6836 2      DSTNAME = DSTPTR[DST$B_NAME];
: 6744      6837 2      DSTNAME[1] = '%';
: 6745      6838 2      CH$`VE(.LENGTH, REGTBL[.REGNUM], DSTNAME[2]);
: 6746      6839 2
: 6747      6840 2
: 6748      6841 2      ! Link the new RST entry into the Temporary RST Entry List.
: 6749      6842 2
: 6750      6843 2      RSTPTR[RST$L_HASH_FLINK] = .RST$TEMP_LIST;
: 6751      6844 2      RST$TEMP_LIST = .RSTPTR;
: 6752      6845 2
: 6753      6846 2
: 6754      6847 2      ! Make a local copy of the Scope List Entry. Then see if the scope is a
: 6755      6848 2      ! module scope (as opposed to a routine scope) and is not explicitly speci-
: 6756      6849 2      ! fied as a pathname; this will commonly be the case in language MACRO.
: 6757      6850 2      ! If so, we change the scope to numeric scope 0 by modifying the local
: 6758      6851 2      ! Scope List Entry. This causes the the current value of the specified
: 6759      6852 2      ! register to be shown.
: 6760      6853 2
: 6761      6854 2      IF .SCOPEPTR EQL 0 THEN RETURN 0;
: 6762      6855 2      CH$MOVE(SCOPE$K_ENTSIZE*%UPVAL, .SCOPEPTR, SCOPEENTRY);
: 6763      6856 2      IF (.SCOPEENTRY[SCOPE$L_STATE] EQL SCOPE$K_NORMAL) AND
: 6764      6857 3      (.PATHDESCR[PTH$B_TOTCNT] EQL 1)
: 6765      6858 2      THEN
: 6766      6859 3      BEGIN
: 6767      6860 3      RPTR = .SCOPEENTRY[SCOPE$L_RSTPTR];
: 6768      6861 3      WHILE .RPTR[RST$B_KIND] NEQ RST$K_ROUTINE DO
: 6769      6862 4      BEGIN
: 6770      6863 4      IF .RPTR[RST$B_KIND] EQL RST$K_MODULE
: 6771      6864 4      THEN
: 6772      6865 5      BEGIN
: 6773      6866 5      SCOPEENTRY[SCOPE$L_STATE] = SCOPE$K_NUMBERED;
: 6774      6867 5      SCOPEENTRY[SCOPE$L_MODPTR] = 0;
: 6775      6868 5      EXITLOOP;
: 6776      6869 4      END;
: 6777      6870 4
: 6778      6871 4      RPTR = .RPTR[RST$L_UPSCOPEPTR];
: 6779      6872 3      END;
: 6780      6873 3
: 6781      6874 2      END;
: 6782      6875 2
: 6783      6876 2
: 6784      6877 2      ! Similarly, if the scope is the Global Scope (the GST) and is not expli-
: 6785      6878 2      ! citly specified as a pathname, we change the scope to numeric scope 0.
: 6786      6879 2      ! This situation is not particularly common, but most likely the user wants
: 6787      6880 2      ! the current register value in this situation.
: 6788      6881 2
: 6789      6882 2      IF (.SCOPEENTRY[SCOPE$L_STATE] EQL SCOPE$K_GLOBAL) AND
: 6790      6883 3      (.PATHDESCR[PTH$B_TOTCNT] EQL 1)
: 6791      6884 2      THEN
```

6792 6885 3
6793 6886 3
6794 6887 3
6795 6888 2
6796 6889 2
6797 6890 2
6798 6891 2
6799 6892 2
6800 6893 2
6801 6894 2
6802 6895 2
6803 6896 2
6804 6897 2
6805 6898 2
6806 6899 2
6807 6900 2
6808 6901 2
6809 6902 2
6810 6903 2
6811 6904 2
6812 6905 3
6813 6906 3
6814 6907 3
6815 6908 3
6816 6909 3
6817 6910 3
6818 6911 3
6819 6912 3
6820 6913 3
6821 6914 3
6822 6915 3
6823 6916 3
6824 6917 3
6825 6918 3
6826 6919 3
6827 6920 4
6828 6921 4
6829 6922 4
6830 6923 5
6831 6924 5
6832 6925 5
6833 6926 5
6834 6927 6
6835 6928 6
6836 6929 6
6837 6930 5
6838 6931 5
6839 6932 5
6840 6933 4
6841 6934 4
6842 6935 3
6843 6936 3
6844 6937 3
6845 6938 3
6846 6939 3
6847 6940 3
6848 6941 3

```
BEGIN
SCOPEENTRY[SCOPE$L_STATE] = SCOPE$K_NUMBERED;
SCOPEENTRY[SCOPE$L_MODPTR] = 0;
END;

: Now determine what scope this register RST entry should be put in. This
: is determined by the kind of the local Scope List Entry.

CASE .SCOPEENTRY[SCOPE$L_STATE] FROM SCOPE$K_NORMAL TO SCOPE$K_SETMODS OF
SET

: Handle normal named scopes. If the scope's module is not set, we
: return zero (no such register). Otherwise, we have a good scope and
: simply put this scope up-scope from the register RST entry. We then
: return the address of the register RST entry as the register SYMID
: unless a line number appeared in the pathname.

[SCOPE$K_NORMAL]:
BEGIN
MODPTR = .SCOPEENTRY[SCOPE$L_MODPTR];
IF NOT .MODPTR[RST$V_MODSET] THEN RETURN 0;
RSTPTR[RST$L_UPSCOPEPTR] = .SCOPEENTRY[SCOPE$L_RSTPTR];

: If there is a line number in the pathname, then the line number
: refers to the lexical entity pointed to by REG_LINE_LEX_PTR. In
: this case we make sure the scope we have contains that entity;
: otherwise the pathname is in error and we return 0. If the
: pathname is okay, we attach the register SYMID to the lexical
: entity specified by the line number.

IF .REG_LINE_LEX_PTR NEQ 0
THEN
BEGIN
RPTR = .REG_LINE_LEX_PTR;
WHILE TRUE DO
BEGIN
IF .RPTR[RST$B_KIND] EQL RST$K_MODULE THEN RETURN 0;
IF .RPTR EQL .RSTPTR[RST$L_UPSCOPEPTR]
THEN
BEGIN
RSTPTR[RST$L_UPSCOPEPTR] = .REG_LINE_LEX_PTR;
EXITLOOP;
END;
RPTR = .RPTR[RST$L_UPSCOPEPTR];
END;
END;

: Unless there was an invocation number in the pathname, return the
: register SYMID now.

IF .PATHDESCR[PTH$B_LOCINVOC] EQL 0 THEN RETURN .RSTPTR;
```

```
6849 6942
6850 6943
6851 6944
6852 6945
6853 6946
6854 6947
6855 6948
6856 6949
6857 6950
6858 6951
6859 6952
6860 6953
6861 6954
6862 6955
6863 6956
6864 6957
6865 6958
6866 6959
6867 6960
6868 6961
6869 6962
6870 6963
6871 6964
6872 6965
6873 6966
6874 6967
6875 6968
6876 6969
6877 6970
6878 6971
6879 6972
6880 6973
6881 6974
6882 6975
6883 6976
6884 6977
6885 6978
6886 6979
6887 6980
6888 6981
6889 6982
6890 6983
6891 6984
6892 6985
6893 6986
6894 6987
6895 6988
6896 6989
6897 6990
6898 6991
6899 6992
6900 6993
6901 6994
6902 6995
6903 6996
6904 6997
6905 6998

! There is an invocation number. Find the inner-most routine con-
! taining the declaration of this symbol. This is the routine to
! which the invocation number must apply.
RPTR = .RSTPTR;
WHILE .RPTR[RST$B_KIND] NEQ RST$K_ROUTINE DO
  BEGIN
    IF .RPTR[RST$B_KIND] EQL RST$K_MODULE
    THEN
      BEGIN
        DBG$NPATHDESC TO CS(.PATHDESCR, PATHSTRING);
        SIGNAL(DBG$_MISIRVNUM, 1, .PATHSTRING);
      END;

    RPTR = .RPTR[RST$L_UPSCOPEPTR];
  END;

! Now make sure the invocation number was indeed appended to that
! routine name in the pathname.
PNAME = .PATHVECC[.PATHDESCR[PTH$B_LOCINVOCC] - 1];
RNAME = DBG$GET_DST_NAME(.RPTR[RST$L_DSTPTR]);
IF CH$NEQ(.PNAME[0], PNAME[1], .RNAME[0], RNAME[1], 0)
THEN
  BEGIN
    DBG$NPATHDESC TO CS(.PATHDESCR, PATHSTRING);
    SIGNAL(DBG$_MISIRVNUM, 1, .PATHSTRING);
  END;

! All looks good. Create the Invocation Number RST Entry along
! with a new copy of the symbol's RST entry if the number is
! non-zero.
IF .PATHDESCR[PTH$L_INVOCNUM] NEQ 0
THEN
  RSTPTR = DBG$BUILD_INVOC_RST(.RSTPTR, .PATHDESCR[PTH$L_INVOCNUM]);

RETURN .RSTPTR;

END;                                ! End of Normal scope code

! Handle numeric scopes. (A 'numeric scope' is the scope a specified
! number of call frames down in the VAX call stack; 2\XR5, for example,
! refers to XR5 in the call frame two levels down in the stack.) Here
! we do one of two things: if we can find a named scope in a SET module
! corresponding to the given numeric scope, we attach the register RST
! entry to that scope; and if we cannot, we create a special 'numeric
! scope' Module RST Entry to represent the unnamed numeric scope. In
! either case, we return a non-zero register SYMID.
[SCOPE$K_NUMBERED]:
  BEGIN
```

```
6906 6999 3
6907 7000 3
6908 7001 3
6909 7002 3
6910 7003 3
6911 7004 3
6912 7005 3
6913 7006 3
6914 7007 3
6915 7008 3
6916 7009 3
6917 7010 4
6918 7011 4
6919 7012 4
6920 7013 4
6921 7014 4
6922 7015 4
6923 7016 4
6924 7017 3
6925 7018 3
6926 7019 3
6927 7020 3
6928 7021 3
6929 7022 3
6930 7023 3
6931 7024 3
6932 7025 3
6933 7026 3
6934 7027 3
6935 7028 3
6936 7029 3
6937 7030 3
6938 7031 3
6939 7032 3
6940 7033 4
6941 7034 4
6942 7035 4
6943 7036 4
6944 7037 4
6945 7038 4
6946 7039 3
6947 7040 3
6948 7041 3
6949 7042 3
6950 7043 3
6951 7044 3
6952 7045 3
6953 7046 3
6954 7047 3
6955 7048 3
6956 7049 3
6957 7050 4
6958 7051 3
6959 7052 3
6960 7053 3
6961 7054 3
6962 7055 3
```

```
See if we can convert this numeric scope to a regular named scope
with a normal RST entry. If so, we put that scope RST entry up-
scope from the register RST entry and return the register SYMID.
(Note that we build an Invocation Number RST Entry if necessary.)
```

```
DBG$STA_NUMBERED_SCOPE(.SCOPEENTRY[SCOPE$L_MODPTR],
                        MODPTR, SCOPE, NUMSCP_INVOC_NUM);
IF .SCOPE NEQ 0
THEN
  BEGIN
    RSTPTR[RST$L_UPSCOPEPTR] = .SCOPE;
    IF .NUMSCP_INVOC_NUM NEQ 0
    THEN
      RSTPTR = DBG$BUILD_INVOC_RST(.RSTPTR, .NUMSCP_INVOC_NUM);

    RETURN .RSTPTR;
  END;
```

```
We have a numeric scope, but it does not correspond to any RST
entry in any SET module. We therefore create a 'numeric scope'
Module RST Entry to represent the numeric scope. This entry has
the RST$V_MODNUMSCP bit set and will therefore be recognized as
representing a numbered stack frame by DBG$STA_SETCONTEXT.
```

```
First generate the name of this pseudo-module, namely the scope
number in Counted ASCII. We get the scope number from the Scope
List Entry.
```

```
VALUE = .SCOPEENTRY[SCOPE$L_MODPTR];
LENGTH = 0;
WHILE TRUE DO
  BEGIN
    NEWVALUE = .VALUE/10;
    NUMTEMP[LENGTH] = .VALUE - .NEWVALUE*10 + '0';
    LENGTH = .LENGTH + 1;
    IF .NEWVALUE EQL 0 THEN EXITLOOP;
    VALUE = .NEWVALUE;
  END;
```

```
NUMTEMP[LENGTH] = .LENGTH;
INCR I FROM 0 TO .LENGTH DO
  NUMNAME[I] = .NUMTEMP[LENGTH - I];
```

```
Now allocate space for this 'module' RST entry and the associated
Module Begin and Module End DST entries. Then fill in the 'num-
eric scope' Module RST Entry, including the RST$V_MODNUMSCP flag.
```

```
MODPTR = DBG$GET_MEMORY(RST$K_MODENTSIZ + (DST$K_MODBEG_SIZE
+ .LENGTH + DST$K_MODEND_SIZE + %UPVAL - 1)%UPVAL);
DSTPTR = .MODPTR + RST$K_MODENTSIZ*%UPVAL;
MODPTR[RST$L_DSTPTR] = .DSTPTR;
MODPTR[RST$B_KIND] = RST$K_MODULE;
MODPTR[RST$B_LANGUAGE] = .DBG$GB_LANGUAGE;
```

```
6963 7056 3
6964 7057 3
6965 7058 3
6966 7059 3
6967 7060 3
6968 7061 3
6969 7062 3
6970 7063 3
6971 7064 3
6972 7065 3
6973 7066 3
6974 7067 3
6975 7068 3
6976 7069 3
6977 7070 3
6978 7071 3
6979 7072 3
6980 7073 3
6981 7074 3
6982 7075 3
6983 7076 3
6984 7077 3
6985 7078 3
6986 7079 3
6987 7080 3
6988 7081 2
6989 7082 2
6990 7083 2
6991 7084 2
6992 7085 2
6993 7086 2
6994 7087 2
6995 7088 2
6996 7089 2
6997 7090 2
6998 7091 2
6999 7092 1
```

```
MODPTR[RST$MODSCPNUM] = .SCOENTRY[SCOPE$L_MODPTR];
MODPTR[RST$MODNUMSCP] = TRUE;
MODPTR[RST$MODSET] = TRUE;
MODPTR[RST$MOD_IN_RST] = TRUE;
```

```
! Also fill in the dummy module's Module Begin and End DST records.
```

```
DSTPTR[DST$B_LENGTH] = DST$K_MODBEG_SIZE - 1 + .LENGTH;
DSTPTR[DST$B_TYPE] = DST$K_MODBEG;
DSTPTR[DST$L_MODBEG_LANGUAGE] = .MODPTR[RST$B_LANGUAGE];
CH$MOVE(.LENGTH + 1, NUMNAME[0], DSTPTR[DST$B_MODBEG_NAME]);
DSTPTR = .DSTPTR + DST$K_MODBEG_SIZE + .LENGTH;
DSTPTR[DST$B_LENGTH] = DST$K_MODEND_SIZE - 1;
DSTPTR[DST$B_TYPE] = DST$K_MODEND;
```

```
! Put the dummy Module RST Entry on the Temporary RST Entry List.
! Also put it up-scope from the register RST entry. Then return
! the address of the register RST entry as the register SYMID.
```

```
MODPTR[RST$L_HASH_FLINK] = .RST$TEMP_LIST;
RST$TEMP_LIST = .MODPTR;
RSTPTR[RST$L_UPSCOPEPTR] = .MODPTR;
RETURN .RSTPTR;
END;
```

```
! Any other scope (such as the global scope) cannot contain a register,
! so we return zero to indicate that this is not a register.
```

```
[INRANGE, OUTRANGE]:
RETURN 0;
```

```
TES;
```

```
END;
```

```
.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
```

```
00441
20 20 30 52 00444 P.AEE: .BLKB 3
20 20 31 52 00448 .ASCII \R0 \
20 20 32 52 0044C .ASCII \R1 \
20 20 33 52 00450 .ASCII \R2 \
20 20 34 52 00454 .ASCII \R3 \
20 20 35 52 00458 .ASCII \R4 \
20 20 36 52 0045C .ASCII \R5 \
20 20 37 52 00460 .ASCII \R6 \
20 20 38 52 00464 .ASCII \R7 \
20 20 39 52 00468 .ASCII \R8 \
20 30 31 52 0046C .ASCII \R9 \
20 31 31 52 00470 .ASCII \R10 \
20 20 50 41 00474 .ASCII \R11 \
20 20 50 46 00478 .ASCII \AP \
                .ASCII \FP \
```

20	20	50	53	0047C	.ASCII	\SP	\
20	20	43	50	00480	.ASCII	\PC	\
20	4C	53	50	00484	.ASCII	\PSL	\
20	32	31	52	00488	.ASCII	\R12	\
20	33	31	52	0048C	.ASCII	\R13	\
20	34	31	52	00490	.ASCII	\R14	\
20	35	31	52	00494	.ASCII	\R15	\

REGTBL=

P.AEE

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

OFFC 00000 GET_REGISTER SYMID:

5B	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	6657
5E		3C	C2	00009	MOVAB	RST\$TEMP_LIST, R11	
5A	04	AC	D0	0000C	SUBL2	#60, SP	
6A	01	AA	91	00010	MOVL	PATHDESCR, R10	6758
		03	13	00014	CMPB	1(R10), (R10)	
		029C	31	00016	BEQL	2\$	
57	08	AA	9E	00019	BRW	34\$	
50	01	AA	9A	0001D	MOVAB	8(R10), PATHVEC	6759
51	FC	A740	D0	00021	MOVZBL	1(R10), R0	6760
56		61	9A	00026	MOVL	-4(PATHVEC)[R0], NAMEPTR	
02		56	D1	00029	MOVZBL	(NAMEPTR), LENGTH	6761
		E8	19	0002C	CMP	LENGTH, #2	6762
04		56	D1	0002E	BLSS	1\$	
		E3	14	00031	CMP	LENGTH, #4	
		51	D6	00033	BGTR	1\$	
25		61	91	00035	INCL	NAMEPTR	6763
		04	12	00038	CMPB	(NAMEPTR), #37	6768
		51	D6	0003A	BNEQ	3\$	
		56	D7	0003C	INCL	NAMEPTR	6771
50		01	CE	0003E	DECL	LENGTH	6772
		17	11	00041	MNEGL	#1, 1	6782
52		5E	C1	00043	BRB	5\$	
		6041	90	00047	ADDL3	SP, 1, R2	
61	8F	62	91	0004B	MOVAB	(1)[NAMEPTR], (R2)	
		09	1F	0004F	CMPB	(R2), #97	6783
7A	8F	62	91	00051	BLSSU	5\$	
		03	1A	00055	CMPB	(R2), #122	
		20	82	00057	BGTRU	5\$	
E5		56	F2	0005A	SUBB2	#32, (R2)	6785
		01	CE	0005E	AOBLSS	LENGTH, 1, 4\$	6780
		54	D4	00061	MNEGL	#1, REGNUM	6793
		44	DF	00063	CLRL	1	6794
03	20	04	AE	00063	PUSHAL	REGTBL[1]	6796
		56	2D	0006A	CMP	LENGTH, TEMPNAME, #32, #3, @ (SP)+	
		9E		00070			
		05	12	00071	BNEQ	7\$	
55		54	D0	00073	MOVL	1, REGNUM	6799
		04	11	00076	BRB	8\$	6798
E7		14	F3	00078	AOBLEQ	#20, 1, 6\$	6794
		55	D5	0007C	TSTL	REGNUM	6808
		96	19	0007E	BLSS	1\$	
10		55	D1	00080	CMP	REGNUM, #16	6814
		03	15	00083	BLEQ	9\$	

		55		05	C2	00085		SUBL2	#5, REGNUM		
		00		0A	DD	00088	9\$:	PUSHL	#10	6819	
	00000000G	59		01	FB	0008A		CALLS	#1, DBG\$GET_MEMORY		
		58	1C	50	D0	00091		MOVL	R0, RSTPTR		
		A9		A9	9E	00094		MOVAB	28(R9), DSTPTR	6820	
	0C	A9		58	D0	00098		MOVL	DSTPTR, 12(RSTPTR)	6821	
	14	A9		06	90	0009C		MOVB	#6, 20(RSTPTR)	6822	
	15	A9	42	8F	88	000A0		BISB2	#6, 21(RSTPTR)	6824	
68		56		08	81	000A5		ADDB3	#8, LENGTH, (DSTPTR)	6831	
	01	A8		08	90	000A9		MOVB	#8, 1(DSTPTR)	6832	
	02	A8		03	88	000AD		BISB2	#3, 2(DSTPTR)	6833	
	03	A8		55	D0	000B1		MOVL	REGNUM, 3(DSTPTR)	6834	
07	A8	56		01	81	000B5		ADDB3	#1, LENGTH, 7(DSTPTR)	6835	
		50	07	A8	9E	000BA		MOVAB	7(DSTPTR), DSTNAME	6836	
	01	A0		25	90	000BE		MOVB	#37, 1(DSTNAME)	6837	
		9E	00000000'EF	45	DF	000C2		PUSHAL	REG\$BL[REGNUM]	6838	
02	A0	69		56	28	000C9		MOVC3	LENGTH, @(\$P)+, 2(DSTNAME)		
		6B		6B	D0	000CE		MOVL	RST\$TEMP_LIST, (RSTPTR)	6843	
				59	D0	000D1		MOVL	RSTPTR, RST\$TEMP_LIST	6844	
			08	AC	D5	000D4		TSTL	SCOPEPTR	6854	
14	AE	08		4F	13	000D7		BEQL	15\$		
		BC		10	28	000D9		MOVC3	#16, @SCOPEPTR, SCOPENTRY	6855	
		01	18	AE	D1	000DF		CMPL	SCOPENTRY+4, #1	6856	
				24	12	000E3		BNEQ	12\$		
		01		6A	91	000E5		CMPB	(R10), #1	6857	
				1F	12	000E8		BNEQ	12\$		
		52	1C	AE	D0	000EA		MOVL	SCOPENTRY+8, RPTR	6860	
		02	14	A2	91	000EE	10\$:	CMPB	20(RPTR), #2	6861	
				15	13	000F2		BEQL	12\$		
		01	14	A2	91	000F4		CMPB	20(RPTR), #1	6863	
				09	12	000F8		BNEQ	11\$		
		18	AE	02	D0	000FA		MOVL	#2, SCOPENTRY+4	6866	
			20	AE	D4	000FE		CLRL	SCOPENTRY+12	6867	
				06	11	00101		BRB	12\$	6865	
		52	10	A2	D0	00103	11\$:	MOVL	16(RPTR), RPTR	6871	
				E5	11	00107		BRB	10\$	6861	
		03	18	AE	D1	00109	12\$:	CMPL	SCOPENTRY+4, #3	6882	
				0C	12	0010D		BNEQ	13\$		
		01		6A	91	0010F		CMPB	(R10), #1	6883	
				07	12	00112		BNEQ	13\$		
		18	AE	02	D0	00114		MOVL	#2, SCOPENTRY+4	6886	
			20	AE	D4	00118		CLRL	SCOPENTRY+12	6887	
0195	03	01	18	AE	CF	0011B	13\$:	CASEL	SCOPENTRY+4, #1, #3	6998	
	0195	00CC		000B		00120	14\$:	.WORD	16\$-14\$,-		
									27\$-14\$,-		
									34\$-14\$,-		
									34\$-14\$		
									34\$	7088	
		10	AE	20	AE	D0	00128	15\$:	BRW		
				10	AE	D0	0012B	16\$:	MOVL	SCOPENTRY+12, MODPTR	6906
				28	A0	E9	00134		MOVL	MODPTR, R0	6907
		10	A9	1C	AE	D0	00138		BLBC	40(R0), 15\$	
				0C	AC	D0	0013D		MOVL	SCOPENTRY+8, 16(RSTPTR)	6908
					1B	13	00141		MOVL	REG_LINE_LEX_PTR, R0	6918
		52		50	D0	00143		BEQL	19\$		
		01	14	A2	91	00146	17\$:	MOVL	R0, RPTR	6921	
				DC	13	0014A		CMPB	20(RPTR), #1	6924	
								BEQL	15\$		

10	A9	52	D1	0014C	CMP	RPTR, 16(RSTPTR)	6925
		06	12	00150	BNEQ	18\$	
10	A9	50	D0	00152	MOVL	R0, 16(RSTPTR)	6928
		06	11	00156	BRB	19\$	6927
	52	10	A2	D0	00158	18\$: MOVL	16(RPTR), RPTR
			E8	11	0015C	BRB	17\$
		02	AA	95	0015E	19\$: TSTB	2(R10)
			77	13	00161	BEQL	24\$
	52		59	D0	00163	MOVL	RSTPTR, RPTR
	02	14	A2	91	00166	20\$: CMPB	20(RPTR), #2
			2A	13	0016A	BEQL	22\$
	01	14	A2	91	0016C	CMPB	20(RPTR), #1
			1E	12	00170	BNEQ	21\$
		04	AE	9F	00172	PUSHAB	PATHSTRING
			5A	DD	00175	PUSHL	R10
00000000G	00		02	FB	00177	CALLS	#2, DBG\$NPATHDESC_TO_CS
		04	AE	DD	0017E	PUSHL	PATHSTRING
			01	DD	00181	PUSHL	#1
		00028C90	8F	DD	00183	PUSHL	#167056
00000000G	00		03	FB	00189	CALLS	#3, LIB\$SIGNAL
	52	10	A2	D0	00190	21\$: MOVL	16(RPTR), RPTR
			D0	11	00194	BRB	20\$
	50	02	AA	9A	00196	22\$: MOVZBL	2(R10), R0
	53	FC	A740	D0	0019A	MOVL	-4(PATHVEC)[R0], PNAME
		0C	A2	DD	0019F	PUSHL	12(RPTR)
00000000G	00		01	FB	001A2	CALLS	#1, DBG\$GET_DST_NAME
	52		63	9A	001A9	MOVZBL	(PNAME), R2
	51		60	9A	001AC	MOVZBL	(RNAME), R1
51	00	01	A3	52	2D	001AF	CMPCS
			01	A0		001B5	
			1E	13	001B7	BEQL	23\$
		04	AE	9F	001B9	PUSHAB	PATHSTRING
			5A	DD	001BC	PUSHL	R10
00000000G	00		02	FB	001BE	CALLS	#2, DBG\$NPATHDESC_TO_CS
		04	AE	DD	001C5	PUSHL	PATHSTRING
			01	DD	001C8	PUSHL	#1
		00028C90	8F	DD	001CA	PUSHL	#167056
00000000G	00		03	FB	001D0	CALLS	#3, LIB\$SIGNAL
		04	AA	D5	001D7	23\$: TSTL	4(R10)
			0D	13	001DA	24\$: BEQL	26\$
		04	AA	DD	001DC	PUSHL	4(R10)
			59	DD	001DF	25\$: PUSHL	RSTPTR
D289	CF		02	FB	001E1	CALLS	#2, DBG\$BUILD_INVOC_RST
	59		50	D0	001E6	MOVL	R0, RSTPTR
		00C5	31	001E9	26\$: BRW	33\$	
			08	AE	9F	001EC	27\$: PUSHAB
			10	AE	9F	001EF	PUSHAB
			18	AE	9F	001F2	PUSHAB
		2C	AE	DD	001F5	PUSHL	SCOPEENTRY+12
E54A	CF		04	FB	001F8	CALLS	#4, DBG\$STA_NUMBERED_SCOPE
		0C	AE	D5	001FD	TSTL	SCOPE
			0F	13	00200	BEQL	28\$
	10	A9	0C	AE	D0	00202	MOVL
			08	AE	D5	00207	TSTL
			DD	13	0020A	BEQL	26\$
		08	AE	DD	0020C	PUSHL	NUMSCP_INVOC_NUM
			CE	11	0020F	BRB	25\$

	52	20	AE	D0	00211	28\$:	MOVL	SCOPENTRY+12, VALUE	7030
			56	D4	00215		CLRL	LENGTH	7031
50	52		0A	C7	00217	29\$:	DIVL3	#10, VALUE, NEWVALUE	7034
51	50		0A	C5	0021B		MULL3	#10, NEWVALUE, R1	7035
	51		52	C2	0021F		SUBL2	VALUE, R1	
24 AE46	30		51	83	00222		SUBB3	R1, #48, NUMTEMP[LENGTH]	
			56	D6	00228		INCL	LENGTH	7036
			50	D5	0022A		TSTL	NEWVALUE	7037
			05	13	0022C		BEQL	30\$	
	52		50	D0	0022E		MOVL	NEWVALUE, VALUE	7038
			E4	11	00231		BRB	29\$	7032
24 AE46			56	90	00233	30\$:	MOVB	LENGTH, NUMTEMP[LENGTH]	7041
51			01	CE	00238		MNEGL	#1, I	7042
			0B	11	0023B		BRB	32\$	
50	56		51	C3	0023D	31\$:	SUBL3	I, LENGTH, R0	7043
F1	30 AE41	24 AE40	90	00241			MOVB	NUMTEMP[R0], NUMNAME[I]	
	51	56	F3	00248	32\$:	AOBLEQ	LENGTH, I, 31\$		
	50	0D	A6	9E	0024C		MOVAB	13(R6), R0	7050
	50		04	C6	00250		DIVL2	#4, R0	7051
		0C	A0	9F	00253		PUSHAB	12(R0)	7050
00000000G	00		01	FB	00256		CALLS	#1, DBG\$GET_MEMORY	
10	AE		50	D0	0025D		MOVL	R0, MODPTR	
	57	10	AE	D0	00261		MOVL	MODPTR, R7	7052
	58	30	A7	9E	00265		MOVAB	48(R7), DSTPTR	
0C	A7		58	D0	00269		MOVL	DSTPTR, 12(R7)	7053
14	A7		01	90	0026D		MOVB	#1, 20(R7)	7054
	50	28	A7	9E	00271		MOVAB	40(R7), R0	7055
01	A0	00000000G	00	90	00275		MOVB	DBG\$GB_LANGUAGE, 1(R0)	
20	A7	20	AE	D0	0027D		MOVL	SCOPENTRY+12, 32(R7)	7056
	60		0B	88	00282		BISB2	#11, (R0)	7059
68	56		07	81	00285		ADDB3	#7, LENGTH, (DSTPTR)	7064
	01	BC	8F	90	00289		MOVB	#-68, 1(DSTPTR)	7065
	03	01	A0	9A	0028E		MOVZBL	1(R0), 3(DSTPTR)	7066
	50	01	A6	9E	00293		MOVAB	1(R6), R0	7067
07 A8	30		50	28	00297		MOVC3	R0, NUMNAME, 7(DSTPTR)	
	58	08 A648	9E	0029D			MOVAB	8(LENGTH)[DSTPTR], DSTPTR	7068
	68	BD01	8F	B0	002A2		MOVW	#48385, (DSTPTR)	7069
	67		6B	D0	002A7		MOVL	RST\$TEMP_LIST, (R7)	7077
	6B		57	D0	002AA		MOVL	R7, RST\$TEMP_LIST	7078
10	A9		57	D0	002AD		MOVL	R7, 16(RSTPTR)	7079
	50		59	D0	002B1	33\$:	MOVL	RSTPTR, R0	7080
				04	002B4		RET		6998
			50	D4	002B5	34\$:	CLRL	R0	7092
				04	002B7		RET		

; Routine Size: 696 bytes, Routine Base: DBG\$CODE + 2C6A

```
7001 7093 1 ROUTINE GET_RECORD_ADDRESS( Primptr, Inner_outer_flag ) =
7002 7094 1
7003 7095 1 FUNCTION
7004 7096 1     GET_RECORD_ADDRESS returns the address of the inner or outer record
7005 7097 1     based on the primary pointer and the flag passed.
7006 7098 1     It's called from the stack machine when the value of a record's field
7007 7099 1     depends on the contents of the record.
7008 7100 1
7009 7101 1 INPUTS
7010 7102 1     Primptr      - A pointer to a primary descriptor passed by value.
7011 7103 1     Inner_outer_flag - A flag indicating whether the inner or outer record
7012 7104 1                   address should be returned.
7013 7105 1
7014 7106 1 OUTPUTS
7015 7107 1     The address of the record.
7016 7108 1
7017 7109 1 SIDE EFFECTS
7018 7110 1     Errors may be signaled
7019 7111 1
7020 7112 2 BEGIN
7021 7113 2
7022 7114 2 LOCAL
7023 7115 2     Current_subnode : REF DBG$PRIM_NODE,
7024 7116 2     Err_vec,
7025 7117 2     Local_current_primary,
7026 7118 2     Local_primary : REF DBG$PRIMARY,
7027 7119 2     Local_val_desc : REF DBG$VALDESC;
7028 7120 2
7029 7121 2     ++
7030 7122 2     Check for a no Primary
7031 7123 2     --
7032 7124 2     IF .Primptr EQLA 0
7033 7125 2     THEN
7034 7126 2         $DBG_ERROR( 'RSTACCESS\GET_RECORD_ADDRESS - zero primary' );
7035 7127 2
7036 7128 2     ++
7037 7129 2     Make a copy of the current primary so we can mangle it
7038 7130 2     --
7039 7131 2     IF NOT DBG$NCOPY_DESC( .Primptr, Local_primary, Err_vec, FALSE )
7040 7132 2     THEN
7041 7133 3         BEGIN
7042 7134 3             EXTERNAL ROUTINE
7043 7135 3                 LIB$SIGNAL : ADDRESSING_MODE(GENERAL);
7044 7136 3             BUILTIN
7045 7137 3                 CALLG;
7046 7138 3             CALLG ( .Err_vec, LIB$SIGNAL );
7047 7139 3             END;
7048 7140 2
7049 7141 2     ++
7050 7142 2     Loop backwards on the primary.
7051 7143 2     1. If we wrap around we know that we want just the first subnode
7052 7144 2        i.e. A.B.C where A is the record whose address we want.
7053 7145 2     2. If we find a TPTR we know we want it and its object.
7054 7146 2        i.e. Q.P^A.B.C where Q.P^ is what we want.
7055 7147 2     3. If we want the inner most record we stop when we see a record
7056 7148 2        subnode.
7057 7149 2     --
```

```

7058 7150 2 Current_subnode = .Local_primary[ DBG$L PRIM BLINK ];
7059 7151 2 WHILE .Current_subnode NEQ Local_primary[ DBG$L PRIM FLINK ] DO
7060 7152 2 BEGIN
7061 7153 2     SELECTONE .Current_subnode[ DBG$B_PNODE_FCODE ] OF
7062 7154 2     SET
7063 7155 2
7064 7156 2     [RST$K_TYPE_RECORD]:
7065 7157 2     IF .Inner_outer_flag EQL Inner
7066 7158 2     THEN
7067 7159 2         EXITLOOP;
7068 7160 2
7069 7161 2     [RST$K_TYPE_PTR,
7070 7162 2     RST$K_TYPE_PTR]:
7071 7163 2     BEGIN
7072 7164 2         Current_subnode = .Current_subnode[ DBG$L_PNODE_FLINK ];
7073 7165 2         EXITLOOP;
7074 7166 2     END;
7075 7167 2
7076 7168 2     TES;
7077 7169 2
7078 7170 2     Current_subnode = .Current_subnode[ DBG$L_PNODE_BLINK ];
7079 7171 2     END;
7080 7172 2
7081 7173 2 ! Go forward one if we wrapped around
7082 7174 2
7083 7175 2 IF .Current_subnode EQL Local_primary[ DBG$L PRIM FLINK ]
7084 7176 2 THEN
7085 7177 2     Current_subnode = .Current_subnode[ DBG$L_PNODE_FLINK ];
7086 7178 2
7087 7179 2 ! Check that all is well
7088 7180 2
7089 7181 2 IF .Current_subnode[ DBG$B_PNODE_FCODE ] NEQ RST$K_TYPE_RECORD
7090 7182 2 THEN
7091 7183 2     $DBG_ERROR( 'RSTACCESS\GET_RECORD_ADDRESS - No record in Primary desc. Bad DST' );
7092 7184 2
7093 7185 2 ! Trim off what we don't want
7094 7186 2
7095 7187 2 Local_primary[ DBG$L PRIM BLINK ] = .Current_subnode;
7096 7188 2 Current_subnode[ DBG$L_PNODE_FLINK ] = Local_primary[ DBG$L PRIM FLINK ];
7097 7189 2
7098 7190 2 ! Save DBG$GL_CURRENT_PRIMARY because DBG$PRIM_TO_VAL updates it
7099 7191 2
7100 7192 2 Local_current_primary = .DBG$GL_CURRENT_PRIMARY;
7101 7193 2
7102 7194 2 ! Get the value
7103 7195 2
7104 7196 2 IF NOT DBG$PRIM_TO_VAL( .Local_primary, DBG$K_V_VALUE_DESC, Local_val_desc )
7105 7197 2 THEN
7106 7198 2     $DBG_ERROR( 'RSTACCESS\GET_RECORD_ADDRESS - DBG$PRIM_TO_VAL failed. Bad DST' );
7107 7199 2
7108 7200 2 ! Restore DBG$GL_CURRENT_PRIMARY because DBG$PRIM_TO_VAL updates it
7109 7201 2
7110 7202 2 DBG$GL_CURRENT_PRIMARY = .Local_current_primary;
7111 7203 2
7112 7204 2 RETURN .Local_val_desc[ DBG$L_VALUE_POINTER ];
7113 7205 2
7114 7206 2 END;
```

```

.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
5F 54 45 47 5C 53 53 45 43 43 41 54 53 52 2B 0049B P.AEF: .ASCII \+RSTACCESS\<92>\GET_RECORD_ADDRESS - ze\
20 53 53 45 52 44 44 41 5F 44 52 4F 43 45 52 004A7
65 7A 20 2D 004B6
79 72 61 6D 69 72 70 20 6F 72 004BA
5F 54 45 47 5C 53 53 45 43 43 41 54 53 52 41 004C4 P.AEG: .ASCII \ro primary\
20 53 53 45 52 44 44 41 5F 44 52 4F 43 45 52 004D3
6D 69 72 50 20 6E 69 20 64 72 6F 63 65 72 20 004E2
44 20 64 61 42 20 2E 63 73 65 64 20 79 72 61 004E6
54 53 00504
5F 54 45 47 5C 53 53 45 43 43 41 54 53 52 3F 00506 P.AEH: .ASCII \?RSTACCESS\<92>\GET_RECORD_ADDRESS - DB\
20 53 53 45 52 44 44 41 5F 44 52 4F 43 45 52 00515
42 41 20 2D 00524
66 20 4C 41 56 5F 4F 54 5F 4D 49 52 5C 24 47 00528
54 53 44 20 64 61 42 20 20 2E 64 65 6C 69 61 00537
.ASCII \G$PRIM_TO_VAL failed. Bad DST\

.EXTRN LIB$SIGNAL
.PSECT DBG$CODE,NOWRT, SHR, PIC,0
```

```

00FC 00000 GET_RECORD_ADDRESS:
57 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7 : 7093
56 00000000' EF 9E 00009 MOVAB DBG$GL_CURRENT_PRIMARY, R7
55 00000000G 00 9E 00010 MOVAB P.AEF, R6
5E 0C C2 00017 MOVAB LIB$SIGNAL, R5
04 AC D5 0001A SUBL2 #12, SP
0D 12 0001D TSTL PRIMPTR : 7124
56 DD 0001F BNEQ 1$
01 DD 00021 PUSHL R6 : 7126
00028362 8F DD 00023 PUSHL #1
65 03 FB 00029 PUSHL #164706
7E D4 0002C 1$: CALLS #3, LIB$SIGNAL : 7131
04 AE 9F 0002E CLRL -(SP)
0C AE 9F 00031 PUSHAB ERR_VEC
04 AC DD 00034 PUSHAB LOCAL_PRIMARY
00 04 04 FB 00037 PUSHL PRIMPTR
04 50 E8 0003E CALLS #4, DBG$NCOPY_DESC
65 00 BE FA 00041 BLBS R0, 2$ : 7138
53 04 AE D0 00045 2$: CALLG @ERR_VEC, LIB$SIGNAL : 7150
52 18 A3 D0 00049 MOVL LOCAL_PRIMARY, R3
54 14 A3 9E 0004D MOVL 24(R3), CURRENT_SUBNODE : 7151
54 52 D1 00051 3$: MOVAB 20(R3), R4
26 13 00054 CMPL CURRENT_SUBNODE, R4
50 09 A2 9A 00056 BEQL 7$ : 7153
07 50 91 0005A MOVZBL 9(CURRENT_SUBNODE), R0 : 7156
08 08 12 0005D CMPB R0, #7
02 08 AC D1 0005F BNEQ 4$ : 7157
11 12 00063 CMPL INNER_OUTER_FLAG, #2
15 11 00065 BNEQ 6$
06 50 91 00067 4$: BRB 7$ : 7159
05 13 0006A CMPB R0, #6 : 7161
BEQL 5$
```

10		50	91	0006C	CMPB	R0, #16	:		
		05	12	0006F	BNEQ	6\$:		
52		62	D0	00071	5\$: MOVL	(CURRENT_SUBNODE), CURRENT_SUBNODE	:	7164	
		06	11	00074	BRB	7\$:	7163	
52	04	A2	D0	00076	6\$: MOVL	4(CURRENT_SUBNODE), CURRENT_SUBNODE	:	7170	
		D5	11	0007A	BRB	3\$:	7151	
54		52	D1	0007C	7\$: CMPL	CURRENT_SUBNODE, R4	:	7175	
		03	12	0007F	BNEQ	8\$:		
52		62	D0	00081	MOVL	(CURRENT_SUBNODE), CURRENT_SUBNODE	:	7177	
07	09	A2	91	00084	8\$: CMPB	9(CURRENT_SUBNODE), #7	:	7181	
		0E	13	00088	BEQL	9\$:		
		2C	A6	9F	0008A	PUSHAB	P.AEG	7183	
		01	DD	0008D	PUSHL	#1	:		
	0002	362	8F	DD	0008F	PUSHL	#164706	:	
18	65		03	FB	00095	CALLS	#3, LIB\$SIGNAL	:	
	A3		52	D0	00098	9\$: MOVL	CURRENT_SUBNODE, 24(R3)	:	7187
	62		54	D0	0009C	MOVL	R4, (CURRENT_SUBNODE)	:	7188
	52		67	D0	0009F	MOVL	DBG\$GL_CURRENT_PRIMARY, -	:	7192
							LOCAL_CURRENT_PRIMARY	:	
		08	AE	9F	000A2	PUSHAB	LOCAL_VAL_DESC	:	7196
	7E	83	8F	9A	000A5	MOVZBL	#131, -(SP)	:	
			53	D0	000A9	PUSHL	R3	:	
00000000G	00		03	FB	000AB	CALLS	#3, DBG\$PRIM_TO_VAL	:	
	0E		50	E8	000B2	BLBS	R0, 10\$:	
		6E	A6	9F	000B5	PUSHAB	P.AEH	:	7198
			01	DD	000B8	PUSHL	#1	:	
	00028362		8F	DD	000BA	PUSHL	#164706	:	
	65		03	FB	000C0	CALLS	#3, LIB\$SIGNAL	:	
	67		52	D0	000C3	10\$: MOVL	LOCAL_CURRENT_PRIMARY, -	:	7202
							DBG\$GL_CURRENT_PRIMARY	:	
	50	08	AE	D0	000C6	MOVL	LOCAL_VAL_DESC, R0	:	7204
	50	18	A0	D0	000CA	MOVL	24(R0), R0	:	
			04	000CE	RET			:	7206

; Routine Size: 207 bytes, Routine Base: DBG\$CODE + 2F22

```
7116 7207 1 ROUTINE GET_REGISTER_VALUES(CURRENT_FP, RUNFRAME_PTR, REGVECTOR): NOVALUE =
7117 7208 1
7118 7209 1 FUNCTION
7119 7210 1 This routine determines the register values associated with a given
7120 7211 1 CALL frame on the VAX call stack. It accepts a PC value and a frame
7121 7212 1 pointer value and some other arguments as input, and produces a vector
7122 7213 1 of register save addresses as output. By indirecting through those
7123 7214 1 save addresses, the actual register values associated with the given
7124 7215 1 CALL frame can be obtained.
7125 7216 1
7126 7217 1 In addition to getting the addresses of all registers saved in a normal
7127 7218 1 CALL frame, this routine understands how to get the register values
7128 7219 1 associated with the CALL frames generated by calls on exception hand-
7129 7220 1 lers (which have a return address pointing into system space) and by
7130 7221 1 DEBUG CALL commands (which have a return address pointing into DEBUG).
7131 7222 1 In the case of calls on exception handlers, some register values (in-
7132 7223 1 cluding the PC) must be gotten from the exception handler's signal and
7133 7224 1 mechanism arguments. In the case of DEBUG CALL commands, all of the
7134 7225 1 register values of the next stack frame must be gotten from DEBUG's
7135 7226 1 stack of saved run-frames.
7136 7227 1
7137 7228 1 INPUTS
7138 7229 1 CURRENT_FP - The address of the CALL frame from which the new set of
7139 7230 1 register save locations is to be extracted. This is thus
7140 7231 1 the value of FP in the called routine.
7141 7232 1
7142 7233 1 RUNFRAME_PTR - The address of a longword which must be initialized to
7143 7234 1 contain the value .DBG$RUNFRAME[DBG$NEXT_LINK] before this
7144 7235 1 routine is called in the course of looping through the
7145 7236 1 CALL stack.
7146 7237 1
7147 7238 1 REGVECTOR - The address of a 17-longword vector to receive all register
7148 7239 1 save addresses from the current CALL frame.
7149 7240 1
7150 7241 1 OUTPUTS
7151 7242 1 RUNFRAME_PTR - The RUNFRAME_PTR location is updated to point to the
7152 7243 1 next saved run-frame on the CALL command run-frame stack
7153 7244 1 each time one such run-frame is accessed to get the register
7154 7245 1 values of the routine which was active at the time of the
7155 7246 1 CALL command.
7156 7247 1
7157 7248 1 REGVECTOR - The addresses at which registers 0 - 16 are saved for the
7158 7249 1 given CALL frame are returned to longwords 0 - 16 of the
7159 7250 1 REGVECTOR vector. (Register 16 is the PSW in this context.)
7160 7251 1
7161 7252 1
7162 7253 2 BEGIN
7163 7254 2
7164 7255 2 MAP
7165 7256 2 CURRENT_FP: REF BLOCK[.BYTE], ! The address of the current CALL frame
7166 7257 2 RUNFRAME_PTR: REF VECTOR[1], ! Pointer to saved-runframe pointer
7167 7258 2 REGVECTOR: REF VECTOR[.LONG]; ! Pointer to the vector of register
7168 7259 2 ! save location addresses
7169 7260 2
7170 7261 2 OWN
7171 7262 2 SPVALUE: REF VECTOR[.LONG]; ! Current CALL frame's SP value
7172 7263 2
```

```

: 7173      7264 2
: 7174      7265 2
: 7175      7266 2
: 7176      7267 2
: 7177      7268 2
: 7178      7269 2
: 7179      7270 2
: 7180      7271 2
: 7181      7272 2
: 7182      7273 2
: 7183      7274 2
: 7184      7275 2
: 7185      7276 2
: 7186      7277 2
: 7187      7278 2
: 7188      7279 2
: 7189      7280 2
: 7190      7281 2
: 7191      7282 2
: 7192      7283 2
: 7193      7284 2
: 7194      7285 2
: 7195      7286 2
: 7196      7287 2
: 7197      7288 2
: 7198      7289 2
: 7199      7290 2
: 7200      7291 2
: 7201      7292 2
: 7202      7293 2
: 7203      7294 2
: 7204      7295 2
: 7205      7296 3
: 7206      7297 3
: 7207      7298 3
: 7208      7299 3
: 7209      7300 3
: 7210      7301 3
: 7211      7302 3
: 7212      7303 3
: 7213      7304 3
: 7214      7305 3
: 7215      7306 3
: 7216      7307 3
: 7217      7308 3
: 7218      7309 3
: 7219      7310 3
: 7220      7311 3
: 7221      7312 4
: 7222      7313 4
: 7223      7314 4
: 7224      7315 5
: 7225      7316 5
: 7226      7317 5
: 7227      7318 4
: 7228      7319 4
: 7229      7320 3

LOCAL
CALLER_PC,
J
MECH_VECTOR: REF BLOCK[.BYTE],
REGMASK: BITVECTOR[16],
REGSAVELOC: REF VECTOR[.LONG],
REGPTR: REF VECTOR[.LONG],
REGVEC: VECTOR[17, .LONG],
SAVED_REGVECTOR:
REF VECTOR[.LONG],
SAVED_RUNFRAME:
REF BLOCK[.BYTE],
SIG_VECTOR: REF VECTOR[.LONG];

: The PC of the caller of the current
: routine (the return address)
: Index value used for several purposes
: Pointer to Mechanism Argument Vector
: The register save mask bit vector
: Pointer to CALL frame register save
: area for registers R0 - R11
: Pointer to a register's save location
: Vector of pointers to save areas for
: Pointer to vector of saved registers
: in saved CALL command runframe
: Pointer to saved runframe from the
: DEBUG CALL command
: Pointer to the Signal Argument Vector

: Get the return PC stored in the current CALL frame.
CALLER_PC = .CURRENT_FP[SFSL_SAVE_PC];

: Check to see if this is an exception handler. (A handler is recognized
: by having a return PC of hex 80000014, which is where the VMS exception
: handling mechanism calls user handlers.) If this is an exception hand-
: ler, we must get the register values of the signaller from the current
: call frame, from the signal arguments, and from the mechanism arguments,
: depending on the register.
IF .CALLER_PC EQL &X'80000014'
THEN
BEGIN

: Extract the save addresses of AP and FP for this CALL frame.
REGVECTOR[12] = CURRENT_FP[SFSL_SAVE_AP];
REGVECTOR[13] = CURRENT_FP[SFSL_SAVE_FP];

: Extract the save locations of all other saved registers in this CALL
: frame.
REGMASK = .CURRENT_FP[SFSW_SAVE_MASK];
REGSAVELOC = CURRENT_FP[SFSL_SAVE_REGS];
J = 0;
INCR J FROM 0 TO 11 DO
BEGIN
IF .REGMASK[J]
THEN
BEGIN
REGVECTOR[J] = REGSAVELOC[J];
J = J + 1;
END;
END;
END;
```



```
7230 7321 3
7231 7322 3
7232 7323 3
7233 7324 3
7234 7325 3
7235 7326 3
7236 7327 3
7237 7328 3
7238 7329 3
7239 7330 3
7240 7331 3
7241 7332 3
7242 7333 3
7243 7334 3
7244 7335 3
7245 7336 3
7246 7337 3
7247 7338 3
7248 7339 3
7249 7340 3
7250 7341 3
7251 7342 3
7252 7343 3
7253 7344 3
7254 7345 3
7255 7346 3
7256 7347 3
7257 7348 3
7258 7349 3
7259 7350 3
7260 7351 3
7261 7352 3
7262 7353 3
7263 7354 3
7264 7355 3
7265 7356 3
7266 7357 3
7267 7358 3
7268 7359 3
7269 7360 3
7270 7361 3
7271 7362 3
7272 7363 3
7273 7364 3
7274 7365 3
7275 7366 3
7276 7367 3
7277 7368 3
7278 7369 3
7279 7370 3
7280 7371 3
7281 7372 3
7282 7373 3
7283 7374 3
7284 7375 3
7285 7376 3
7286 7377 3

! Set the stack pointer to point at the end of the saved registers.
! Adjust it by the offset value. Also pass the one longword of junk
! the VMS signal mechanism puts on the stack (a JSB return address).
SPVALUE = REGSAVELOC[J];
SPVALUE = .SPVALUE + .CURRENT_FP[SFSV_STACKOFFS];
SPVALUE = .SPVALUE + 4;

! Get the pointer to the signal argument list and pick up the address
! of the saved PC in the signal argument list. We also pick up the
! address of the saved PSL in the signal argument list.
SIG_VECTOR = .SPVALUE[1];
J = .SIG_VECTOR[0];
REGVECTOR[15] = SIG_VECTOR[J - 1];
REGVECTOR[16] = SIG_VECTOR[J];

! Get the pointer to the mechanism argument list and pick up the save
! addresses of the signaller's values of R0 and R1.
MECH_VECTOR = .SPVALUE[2];
REGVECTOR[0] = MECH_VECTOR[CHFSL_MCH_SAVR0];
REGVECTOR[1] = MECH_VECTOR[CHFSL_MCH_SAVR1];

! Finally compute the SP value by skipping past the exception handler
! argument list, the list of signal arguments, one longword of trash,
! and the list of mechanism arguments.
SPVALUE = SPVALUE[.SPVALUE[0] + 1];
SPVALUE = SPVALUE[.SPVALUE[0] + 1];
SPVALUE = .SPVALUE + 4;
SPVALUE = SPVALUE[.SPVALUE[0] + 1];
REGVECTOR[14] = SPVALUE;
END

! Check to see if the current routine was called with a DEBUG CALL command.
! (A CALL command is recognized by the DBG$PSEUDO_EXIT return address.)
! If so, we must dig out all the register values as they were at the time
! of the CALL. We dig out the save locations of these values from the run-
! frame at our current location on the saved-runframe stack.
ELSE IF .CALLER_PC EQL DBG$PSEUDO_EXIT
THEN
BEGIN
  SAVED_RUNFRAME = .RUNFRAME_PTR[0];
  SAVED_REGVECTOR = SAVED_RUNFRAME[DBG$SL_USER_R0];
  INCR I FROM 0 TO 16 DO
    REGVECTOR[I] = SAVED_REGVECTOR[I];

  RUNFRAME_PTR[0] = .SAVED_RUNFRAME[DBG$SL_NEXT_LINK];
END
```

```
7287 7378 3
7288 7379 3
7289 7380 3
7290 7381 3
7291 7382 3
7292 7383 3
7293 7384 3
7294 7385 3
7295 7386 3
7296 7387 3
7297 7388 3
7298 7389 3
7299 7390 3
7300 7391 3
7301 7392 3
7302 7393 3
7303 7394 4
7304 7395 4
7305 7396 4
7306 7397 5
7307 7398 5
7308 7399 5
7309 7400 4
7310 7401 4
7311 7402 3
7312 7403 3
7313 7404 3
7314 7405 3
7315 7406 3
7316 7407 3
7317 7408 3
7318 7409 3
7319 7410 3
7320 7411 3
7321 7412 3
7322 7413 3
7323 7414 3
7324 7415 3
7325 7416 3
7326 7417 3
7327 7418 3
7328 7419 3
7329 7420 3
7330 7421 3
7331 7422 3
7332 7423 3
7333 7424 3
7334 7425 3
7335 7426 3
7336 7427 3
7337 7428 3
7338 7429 3
7339 7430 3
7340 7431 3
7341 7432 3
7342 7433 3
7343 7434 2
```

```
! For any other case, we have a normal CALL frame on the stack and we can
! dig out the register values in the normal way. That is done here.
```

```
ELSE
  BEGIN
```

```
! Get the save locations of all registers of the set R0 - R11 that are
! saved in this CALL frame. Save those addresses in REGVECTOR.
```

```
REGMASK = .CURRENT_FP[SF$W_SAVE_MASK];
REGSAVELOC = CURRENT_FP[SF$SL_SAVE_REGS];
```

```
J = 0;
INCR I FROM 0 TO 11 DO
```

```
  BEGIN
```

```
    IF .REGMASK[I]
```

```
    THEN
```

```
      BEGIN
```

```
        REGVECTOR[I] = REGSAVELOC[J];
```

```
        J = J + 1;
```

```
      END;
```

```
  END;
```

```
! If R0 or R1 is not saved, we zero the corresponding REGVECTOR cells
! to indicate that those registers are not available at all--R0 and R1
! are not preserved over subroutine calls.
```

```
IF NOT .REGMASK[0] THEN REGVECTOR[0] = 0;
```

```
IF NOT .REGMASK[1] THEN REGVECTOR[1] = 0;
```

```
! Get the addresses of the save locations for registers AP, FP, SP,
! PC, and PSW. Store those addresses in REGVECTOR.
```

```
REGVECTOR[12] = CURRENT_FP[SF$SL_SAVE_AP];
```

```
REGVECTOR[13] = CURRENT_FP[SF$SL_SAVE_FP];
```

```
REGVECTOR[14] = SPVALUE;
```

```
REGVECTOR[15] = CURRENT_FP[SF$SL_SAVE_PC];
```

```
REGVECTOR[16] = CURRENT_FP[SF$SW_SAVE_PSW];
```

```
! Determine the value of SP (the Stack Pointer) by pointing it to the
! end of the register save area, adjusting it by the offset value, and
! pointing it past the CALLS argument list (if any). Save the computed
! SP value in SPVALUE.
```

```
SPVALUE = REGSAVELOC[J];
```

```
SPVALUE = SPVALUE + .CURRENT_FP[SF$SV_STACKOFFS];
```

```
IF .CURRENT_FP[SF$SV_CALLS] THEN SPVALUE = SPVALUE + 4*(.SPVALUE[0] + 1);
```

```
END;
```

```
! We are done getting the register values and can now return.
```

			58	00000000'	EF	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8	: 7207
			5E		BC	AE	9E	00009	SPVALUE, R8	:
			56		04	AC	D0	0000D	-68(SP), SP	:
			51		10	A6	D0	00011	CURRENT_FP, R6	: 7284
			53		0C	AC	D0	00015	16(R6), -CALLER_PC	:
		80000014	8F			51	D1	00019	REGVECTOR, R3	: 7301
						7F	12	00020	CALLER_PC, #-2147483628	: 7294
						55	D4	00034	3\$:
		30	A3	08	A6	9E	00022	MOVAB	8(R6), 48(R3)	: 7301
		34	A3	0C	A6	9E	00027	MOVAB	12(R6), 52(R3)	: 7302
			57	06	A6	B0	0002C	MOVW	6(R6), REGMASK	: 7308
			51	14	A6	9E	00030	MOVAB	20(R6), REGSAVELOC	: 7309
						50	D4	00036	J	: 7310
						50	E1	00038	I	: 7311
	07		57					BBC	I, REGMASK, 2\$: 7313
		6340			6145	DE	0003C	MOVAL	(REGSAVELOC)[J], (R3)[I]	: 7316
						55	D6	00041	J	: 7317
	F1		50		0B	F3	00043	AOBLEQ	#11, I, 1\$: 7311
			68		6145	DE	00047	MOVAL	(REGSAVELOC)[J], SPVALUE	: 7327
50	07	A6	02		06	EF	0004B	EXTZV	#6, #2, 7(R6), R0	: 7328
			68		50	C0	00051	ADDL2	R0, SPVALUE	:
			68		04	C0	00054	ADDL2	#4, SPVALUE	: 7329
			50		68	D0	00057	MOVL	SPVALUE, R0	: 7336
			52	04	A0	D0	0005A	MOVL	4(R0), SIG VECTOR	:
			55		62	D0	0005E	MOVL	(SIG VECTOR), J	: 7337
	3C	A3		FC	A245	DE	00061	MOVAL	-4(SIG VECTOR)[J], 60(R3)	: 7338
	40	A3			6245	DE	00067	MOVAL	(SIG VECTOR)[J], 64(R3)	: 7339
		51		08	A0	D0	0006C	MOVL	8(R0), MECH VECTOR	: 7345
		63		0C	A1	9E	00070	MOVAB	12(R1), (R3)	: 7346
	04	A3		10	A1	9E	00074	MOVAB	16(R1), 4(R3)	: 7347
		50			60	D0	00079	MOVL	(R0), R0	: 7354
		78			9840	DE	0007C	MOVAL	@SPVALUE[R0], SPVALUE	:
		68			04	C0	00080	ADDL2	#4, SPVALUE	:
		50		00	B8	D0	00083	MOVL	@SPVALUE, R0	: 7355
		78			9840	DE	00087	MOVAL	@SPVALUE[R0], SPVALUE	:
		68			04	C0	0008B	ADDL2	#4, SPVALUE	:
		68			04	C0	0008E	ADDL2	#4, SPVALUE	: 7356
		50		00	B8	D0	00091	MOVL	@SPVALUE, R0	: 7357
		78			9840	DE	00095	MOVAL	@SPVALUE[R0], SPVALUE	:
		68			04	C0	00099	ADDL2	#4, SPVALUE	:
	38	A3			68	9E	0009C	MOVAB	SPVALUE, 56(R3)	: 7358

RST
V04

Label	Offset	Hex	Asm	Comment	Address
	50	00000000G	00 04 000A0	RET	7294
	50		51 9E 000A1 3\$:	MOVAB DBG\$PSEUDO_EXIT, R0	7368
			18 D1 000A8	CMPL CALLER_PC, R0	
	50	08	12 000AB	BNEQ 5\$	
	52	04	BC D0 000AD	MOVL @RUNFRAME_PTR, SAVED_RUNFRAME	7371
			A0 9E 000B1	MOVAB 4(R0), SAVED_REGVECTOR	7372
			54 D4 000B5	CLRL I	7374
F7	6344		DE 000B7 4\$:	MOVAL (SAVED_REGVECTOR)[I], (R3)[I]	
	54		10 F3 000BC	AOBLEQ #16, I, 4\$	
	08	BC	60 D0 000C0	MOVL (SAVED_RUNFRAME), @RUNFRAME_PTR	7376
			04 000C4	RET	7368
	57	06	B0 000C5 5\$:	MOVW 6(R6), REGMASK	7390
	51	14	A6 9E 000C9	MOVAB 20(R6), REGSAVELOC	7391
			55 D4 000CD	CLRL J	7392
			50 D4 000CF	CLRL I	7393
08	57		50 E1 000D1 6\$:	BBC I, REGMASK, 7\$	7395
	0C	BC40	6145 DE 000D5	MOVAL (REGSAVELOC)[J], @REGVECTOR[I]	7398
			55 D6 000DB	INCL J	7399
F0	50		08 F3 000DD 7\$:	AOBLEQ #11, I, 6\$	7393
	03		57 E8 000E1	BLBS REGMASK, 8\$	7409
		0C	BC D4 000E4	CLRL @REGVECTOR	
07	57		01 E0 000E7 8\$:	BBS #1, REGMASK, 9\$	7410
	50	0C	AC D0 000EB	MOVL REGVECTOR, R0	
		04	A0 D4 000EF	CLRL 4(R0)	
	30	A3	08 A6 9E 000F2 9\$:	MOVAB 8(R6), 48(R3)	7416
	34	A3	0C A6 9E 000F7	MOVAB 12(R6), 52(R3)	7417
	38	A3	68 9E 000FC	MOVAB SPVALUE, 56(R3)	7418
	3C	A3	10 A6 9E 00100	MOVAB 16(R6), 60(R3)	7419
	40	A3	04 A6 9E 00105	MOVAB 4(R6), 64(R3)	7420
A6	68		6145 DE 0010A	MOVAL (REGSAVELOC)[J], SPVALUE	7428
	02		06 EF 0010E	EXITV #6, #2, 7(R6), R0	7429
	68		50 C0 00114	ADDL2 R0, SPVALUE	
0B	07		05 E1 00117	BBC #5, 7(R6), 10\$	7430
		00	B8 D0 0011C	MOVL @SPVALUE, R0	
			9840 DE 00120	MOVAL @SPVALUE[R0], SPVALUE	
			04 C0 00124	ADDL2 #4, SPVALUE	
			04 00127 10\$:	RET	7438

```
; Routine Size: 296 bytes,   Routine Base: DBG$CODE + 2FF1
```

```
7349 7439 1 ROUTINE SCOPE_RULE_COBOL(PATHNAME, NCANDS, CANDLST, SCOPE) =
7350 7440 1
7351 7441 1 FUNCTION
7352 7442 1 This routine selects the symbol from a specified list of candidate sym-
7353 7443 1 bols which best matches a specified pathname. This routine assumes
7354 7444 1 COBOL scope rules when doing so. This means that incomplete data quali-
7355 7445 1 fication is accepted, and that uniqueness is determined by these rules:
7356 7446 1
7357 7447 1 (1) By definition, the "lowest definition depth" is the
7358 7448 1 inner-most definition depth in the current scope at
7359 7449 1 which at least one candidate symbol is declared.
7360 7450 1
7361 7451 1 (2) If only one candidate symbol is defined at the lowest
7362 7452 1 definition depth, then that is the unique symbol we
7363 7453 1 want.
7364 7454 1
7365 7455 1 (3) Otherwise, the symbol is not unique.
7366 7456 1
7367 7457 1 An additional COBOL scope rule is that any candidate which is not marked
7368 7458 1 as "global" (i.e., does not have the RSTSV_COBOLGBL bit set) may not be
7369 7459 1 declared outside the routine which contains the current scope. In other
7370 7460 1 words, a COBOL symbol declared in one routine is not visible in any
7371 7461 1 nested routine unless it is specifically marked as being so visible.
7372 7462 1
7373 7463 1 The list of candidate symbols is produced by DBG$STA_GETSYMBOL, and each
7374 7464 1 candidate is guaranteed to be in the current scope being searched. What
7375 7465 1 this routine must do is to determine which candidates have valid data
7376 7466 1 qualification, which candidate is defined at the lowest definition depth
7377 7467 1 (i.e., defined inner-most in the current scope), and whether that candi-
7378 7468 1 date is unique. The routine then returns one of three things: an indi-
7379 7469 1 cation that no symbol was valid, an indication that the symbol is not
7380 7470 1 unique, or an index pointing to the one selected candidate symbol.
7381 7471 1
7382 7472 1 INPUTS
7383 7473 1 PATHNAME - Pointer to the pathname descriptor for the symbol name to
7384 7474 1 be looked up in the symbol table.
7385 7475 1
7386 7476 1 NCANDS - The number of candidate symbols found by DBG$STA_GETSYMBOL.
7387 7477 1
7388 7478 1 CANDLST - A vector of pointers to the "candidate blocks" for the candi-
7389 7479 1 date symbols found by DBG$STA_GETSYMBOL. Each of these candi-
7390 7480 1 dates is in the scope currently searched. The candidate block
7391 7481 1 pointers are found in CANDLST[1] through CANDLST[NCANDS].
7392 7482 1
7393 7483 1 SCOPE - A pointer to the RST entry for the current scope in which the
7394 7484 1 symbol is being looked up. This normally points to a Routine
7395 7485 1 RST Entry or a Lexical Block RST Entry. (COBOL Sections and
7396 7486 1 Paragraphs are represented as lexical blocks in DEBUG.) If
7397 7487 1 the current scope is the Global Scope (\) or All Set Modules,
7398 7488 1 the SCOPE parameter is zero.
7399 7489 1
7400 7490 1 OUTPUTS
7401 7491 1 The CANDLST index for the candidate block which best matches the path-
7402 7492 1 name is returned as the routine's value. If no candidate is
7403 7493 1 acceptable, zero is returned, and if more than one candidate
7404 7494 1 is acceptable (the symbol is not unique), -1 is returned.
7405 7495 1
```

```

: 7406      7496      1
: 7407      7497      2
: 7408      7498      2
: 7409      7499      2
: 7410      7500      2
: 7411      7501      2
: 7412      7502      2
: 7413      7503      2
: 7414      7504      2
: 7415      7505      2
: 7416      7506      2
: 7417      7507      2
: 7418      7508      2
: 7419      7509      2
: 7420      7510      2
: 7421      7511      2
: 7422      7512      2
: 7423      7513      2
: 7424      7514      2
: 7425      7515      2
: 7426      7516      2
: 7427      7517      2
: 7428      7518      2
: 7429      7519      2
: 7430      7520      2
: 7431      7521      2
: 7432      7522      2
: 7433      7523      2
: 7434      7524      2
: 7435      7525      2
: 7436      7526      2
: 7437      7527      2
: 7438      7528      2
: 7439      7529      2
: 7440      7530      2
: 7441      7531      2
: 7442      7532      2
: 7443      7533      2
: 7444      7534      2
: 7445      7535      2
: 7446      7536      2
: 7447      7537      3
: 7448      7538      3
: 7449      7539      3
: 7450      7540      3
: 7451      7541      3
: 7452      7542      3
: 7453      7543      3
: 7454      7544      4
: 7455      7545      4
: 7456      7546      4
: 7457      7547      4
: 7458      7548      4
: 7459      7549      4
: 7460      7550      4
: 7461      7551      4
: 7462      7552      4

BEGIN
MAP
    PATHNAME: REF PTH$PATHNAME,      ! Pointer to symbol pathname descriptor
    CANDLST: REF VECTOR[,LONG],      ! Pointer to candidate vector
    SCOPE: REF RST$ENTRY;             ! Pointer to current scope RST entry
LABEL
    CHECK_THIS_CANDIDATE;             ! Label of block we want to LEAVE
LOCAL
    CANDBLK: REF CAND_BLOCKVECTOR,    ! Pointer to current "candidate block"
    COBOLGBL_FLAG,                    ! Flag set to TRUE if the COBOL Global
                                     ! Attribute applies to this item
    DATA_INDEX,                      ! Index into CANDBLK vector of Data Item
                                     ! RST Entry pointer (or zero)
    DATAQUAL_FLAG,                  ! Set to TRUE when we are in the data
                                     ! qualification part of a name
    DEFDEPTH,                        ! Definition depth of current candidate
    DSTPTR: REF DST$RECORD,           ! Pointer to symbol DST record
    GOOD_CAND,                       ! CANDLST index of best candidate so far
    GOOD_DEFDEPTH,                   ! Definition depth of GOOD_CAND symbol
    J,                               ! Index for CANDBLK vector
    RSTPTR: REF RST$ENTRY,            ! Pointer to current symbol RST entry
    SCPTR: REF RST$ENTRY,             ! Pointer used to follow current scope's
                                     ! up-scope chain
    SYMSCOPE: REF RST$ENTRY;          ! The actual scope of the current symbol

! Initially we do not have a good candidate.
GOOD_CAND = 0;
GOOD_DEFDEPTH = 1000000;

! Loop over all the candidate blocks on the candidate list. This loop
! searches for the best candidate symbol matching the pathname.
INCR I FROM 1 TO .NCANDS DO
    BEGIN

        ! Set up a labelled block to check out the current candidate. We can
        ! LEAVE this block if we find that the candidate is not acceptable.
        CHECK_THIS_CANDIDATE:
        BEGIN
            CANDBLK = .CANDLST[I];

            ! Loop over the candidate's up-scope chain--that is what the CANDBLK
            ! vector gives us. Reject any candidate whose data qualification in
            ! the up-scope chain does not agree with that in the pathname.
            DATA_INDEX = 0;
```

```
: 7463      7553      4      COBOLGBL_FLAG = FALSE;
: 7464      7554      4      DATAQUAL_FLAG = TRUE;
: 7465      7555      4      J = 0;
: 7466      7556      4      WHILE .CANDBLK[J, CAND_RSTPTR] NEQ 0 DO
: 7467      7557      5          BEGIN
: 7468      7558      5              RSTPTR = .CANDBLK[J, CAND_RSTPTR];
: 7469      7559      5
: 7470      7560      5
: 7471      7561      5              ! Clear DATAQUAL_FLAG if we have left the data qualification
: 7472      7562      5              ! part of the name.
: 7473      7563      5
: 7474      7564      5              IF (.CANDBLK[J, CAND_PINDEX] LSS .PATHNAME[PTH$B_PATHCNT]) AND
: 7475      7565      6                  (.CANDBLK[J, CAND_PINDEX] NEQ 0)
: 7476      7566      5              THEN
: 7477      7567      5                  DATAQUAL_FLAG = FALSE;
: 7478      7568      5
: 7479      7569      5              IF (.RSTPTR[RST$B_KIND] NEQ RST$K_DATA) AND
: 7480      7570      6                  (.RSTPTR[RST$B_KIND] NEQ RST$K_TYPCOMP)
: 7481      7571      5              THEN
: 7482      7572      5                  DATAQUAL_FLAG = FALSE;
: 7483      7573      5
: 7484      7574      5
: 7485      7575      5              ! After we leave the data qualification part going up-scope, we
: 7486      7576      5              ! do not accept Data Items or Type Components in the name.
: 7487      7577      5
: 7488      7578      5              IF (NOT .DATAQUAL_FLAG) AND
: 7489      7579      6                  (.RSTPTR[RST$B_KIND] EQL RST$K_DATA OR
: 7490      7580      6                  .RSTPTR[RST$B_KIND] EQL RST$K_TYPCOMP)
: 7491      7581      5              THEN
: 7492      7582      5                  LEAVE CHECK_THIS_CANDIDATE;
: 7493      7583      5
: 7494      7584      5
: 7495      7585      5              ! If this is the main Data Item RST Entry in the CANDBLK up-
: 7496      7586      5              ! scope chain, save its index in DATA_INDEX. Also set the
: 7497      7587      5              ! COBOL Global Attribute flag if the RST entry is so marked.
: 7498      7588      5
: 7499      7589      5              IF .RSTPTR[RST$B_KIND] EQL RST$K_DATA
: 7500      7590      5              THEN
: 7501      7591      6                  BEGIN
: 7502      7592      6                      DATA_INDEX = .J;
: 7503      7593      6                      IF .RSTPTR[RST$V_COBOLGBL] THEN COBOLGBL_FLAG = TRUE;
: 7504      7594      5                      END;
: 7505      7595      5
: 7506      7596      5
: 7507      7597      5              ! Increment the CANDBLK index and loop up-scope.
: 7508      7598      5
: 7509      7599      5              J = .J + 1;
: 7510      7600      4              END;
: 7511      7601      4
: 7512      7602      4
: 7513      7603      4              ! Pick up the definition depth from the last CANDBLK cell. Reject
: 7514      7604      4              ! this candidate if we already have a candidate with a smaller def-
: 7515      7605      4              ! inition depth (i.e., defined closer to the current scope).
: 7516      7606      4
: 7517      7607      4              RSTPTR = .CANDBLK[DATA_INDEX, CAND_RSTPTR];
: 7518      7608      4              DEFDEPTH = .CANDBLK[J, CAND_PINDEX];
: 7519      7609      4              IF .DEFDEPTH GTR .GOOD_DEFDEPTH THEN LEAVE CHECK_THIS_CANDIDATE;
```

: 7520 7610 4
: 7521 7611 4
: 7522 7612 4
: 7523 7613 4
: 7524 7614 4
: 7525 7615 4
: 7526 7616 4
: 7527 7617 4
: 7528 7618 4
: 7529 7619 4
: 7530 7620 4
: 7531 7621 4
: 7532 7622 5
: 7533 7623 4
: 7534 7624 5
: 7535 7625 5
: 7536 7626 5
: 7537 7627 5
: 7538 7628 5
: 7539 7629 5
: 7540 7630 5
: 7541 7631 5
: 7542 7632 5
: 7543 7633 5
: 7544 7634 5
: 7545 7635 5
: 7546 7636 5
: 7547 7637 5
: 7548 7638 5
: 7549 7639 5
: 7550 7640 5
: 7551 7641 5
: 7552 7642 5
: 7553 7643 5
: 7554 7644 5
: 7555 7645 5
: 7556 7646 6
: 7557 7647 6
: 7558 7648 6
: 7559 7649 6
: 7560 7650 6
: 7561 7651 6
: 7562 7652 6
: 7563 7653 6
: 7564 7654 6
: 7565 7655 6
: 7566 7656 5
: 7567 7657 5
: 7568 7658 4
: 7569 7659 4
: 7570 7660 4
: 7571 7661 4
: 7572 7662 4
: 7573 7663 4
: 7574 7664 4
: 7575 7665 4
: 7576 7666 4

: Unless the COBOL 'global' flag is set for this symbol, we see if
: the symbol is declared in a routine outside the current scope.
: If it is, we must reject the symbol. In COBOL, a symbol is not
: visible in nested routines unless marked as 'global'. Note that
: we skip this check if SCOPE is zero, meaning that the scope is
: the GST or all SET modules. We also skip the check for symbols
: which are not data--these rules do not apply to routines, etc.

IF (NOT .COBOLGBL_FLAG) AND
 (.SCOPE NEQ 0) AND
 (.RSTPTR[RST\$B_KIND] EQL RST\$K_DATA)
THEN
 BEGIN

: Determine the scope in which the current symbol is declared.

SYMSCOPE = .RSTPTR;
IF .RSTPTR[RST\$B_KIND] NEQ RST\$K_MODULE
THEN
 SYMSCOPE = .RSTPTR[RST\$L_UPSCOPEPTR];

IF .SYMSCOPE[RST\$B_KIND] EQL RST\$K_TYPE
THEN
 SYMSCOPE = .SYMSCOPE[RST\$L_UPSCOPEPTR];

: See if there is a routine declaration between the current
: scope and the environment in which the symbol is declared.
: If so, reject this candidate--it is not visible from the
: current scope.

SCPTR = .SCOPE;
WHILE .SCPTR NEQ .SYMSCOPE DO
 BEGIN
 IF .SCPTR[RST\$B_KIND] EQL RST\$K_ROUTINE
 THEN
 LEAVE CHECK_THIS_CANDIDATE;

 IF .SCPTR[RST\$B_KIND] EQL RST\$K_MODULE
 THEN
 \$DBG_ERROR('RSTACCESS\SCOPE_RULE_COBOL');

 SCPTR = .SCPTR[RST\$L_UPSCOPEPTR];
 END;

END;

: We have a good candidate here. If we already have another candi-
: date at the same definition depth, the symbol maybe is not unique.
: We call a routine which attempts to resolve the amiguity. If it
: resolves the ambiguity, then it returns the appropriate index.
: It returns -1 if the reference really is amiguous.

! Return the GOOD_CAND value. This may be -1, 0, or a true CANDLIST index.

RETURN .GOOD_CAND;

END:

```
.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
```

```
50  4F  43  53  5C  53  53  45  43  43  41  54  53  52  1A  00546 P.AEI:  .ASCII  <26>\RSTACCESS\<92>\SCOPE_RULE_COBOL\
      4C  4F  42  4F  43  5F  45  4C  55  52  5F  45  00555
```

```
.PSECT DBGS$CODE,NOWRT, SHR, PIC,0
```

OFFC 00000 SCOPE_RULE COBOL:									
5E	08	C2	00002		.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	:	7439	
	59	D4	00005		SUBL2	#8, SP	:		
5A	000F4240	8F	D0	00007	CLRL	GOOD CAND	:	7529	
	56	D4	0000E		MOVL	#1000000, GOOD_DEFDEPTH	:	7530	
	00F2	31	D0	00010	CLRL	I	:	7564	
54	0C	BC46	D0	00013	BRW	15\$:		
	58	D4	00018	2\$:	MOVL	@CANDLST[I], CANDBLK	:	7545	
	6E	D4	0001A		CLRL	DATA INDEX	:	7552	
04	AE	01	D0	0001C	CLRL	COBO[GBL_FLAG	:	7553	
	52	D4	00020		MOVL	#1, DATAQUAL_FLAG	:	7554	
	6442	7F	00022	3\$:	CLRL	J	:	7555	
	9E	D5	00025		PUSHAQ	(CANDBLK)[J]	:	7556	
	50	13	00027		TSTL	@(SP)+	:		
	6442	7F	00029		BEQL	8\$:		
53		9E	D0	0002C	PUSHAQ	(CANDBLK)[J]	:	7558	
	04	A442	7F	0002F	MOVL	@(SP)+, RSTPTR	:		
					PUSHAQ	4(CANDBLK)[J]	:	7564	

9E	04	BC	08	08	ED	00033	CMPZV	#8, #8, @PATHNAME, @ (SP)+		
				0B	15	00039	BLEQ	4\$		
				04	A442	7F	0003B	PUSHAQ	4(CANDBLK)[J]	7565
				9E	D5	0003F	TSTL	@ (SP)+		
				03	13	00041	BEQL	4\$		
				04	AE	D4	00043	CLRL	DATAQUAL_FLAG	7567
			50	14	A3	9A	00046	MOVZBL	20(RSTPTR), R0	7569
			06	50	91	0004A	CMPB	R0, #6		
				08	13	0004D	BEQL	5\$		
			0A	50	91	0004F	CMPB	R0, #10	7570	
				03	13	00052	BEQL	5\$		
				04	AE	D4	00054	CLRL	DATAQUAL_FLAG	7572
			0A	04	AE	E8	00057	BLBS	DATAQUAL_FLAG, 6\$	7578
			06	50	91	0005B	CMPB	R0, #6	7579	
				B0	13	0005E	BEQL	1\$		
			0A	50	91	00060	CMPB	R0, #10	7580	
				AB	13	00063	BEQL	1\$		
			06	50	91	00065	CMPB	R0, #6	7589	
				08	12	00068	BNEQ	7\$		
			58	52	D0	0006A	MOVL	J, DATA_INDEX	7592	
	03	15	A3	05	E1	0006D	BBC	#5, 21(RSTPTR), 7\$	7593	
			6E	01	D0	00072	MOVL	#1, COBOLGBL_FLAG		
				52	D6	00075	INCL	J	7599	
				A9	11	00077	BRB	3\$	7556	
				6448	7F	00079	PUSHAQ	(CANDBLK)[DATA_INDEX]	7607	
			53	9E	D0	0007C	MOVL	@ (SP)+, RSTPTR		
				04	A442	7F	0007F	PUSHAQ	4(CANDBLK)[J]	7608
			5B	9E	D0	00083	MOVL	@ (SP)+, DEFDEPTH		
			5A	5B	D1	00086	CMPL	DEFDEPTH, GOOD_DEFDEPTH	7609	
				7A	14	00089	BGTR	15\$		
			52	6E	E8	0008B	BLBS	COBOLGBL_FLAG, 13\$	7620	
				10	AC	D5	0008E	TSTL	SCOPE	7621
				4D	13	00091	BEQL	13\$		
			06	14	A3	91	00093	CMPB	20(RSTPTR), #6	7622
				47	12	00097	BNEQ	13\$		
			57	53	D0	00099	MOVL	RSTPTR, SYMSCOPE	7629	
			01	14	A3	91	0009C	CMPB	20(RSTPTR), #1	7630
				04	13	000A0	BEQL	9\$		
			57	10	A3	D0	000A2	MOVL	16(RSTPTR), SYMSCOPE	7632
			07	14	A7	91	000A6	CMPB	20(SYMSCOPE), #7	7634
				04	12	000AA	BNEQ	10\$		
			57	10	A7	D0	000AC	MOVL	16(SYMSCOPE), SYMSCOPE	7636
			55	10	AC	D0	000B0	MOVL	SCOPE, SCPTR	7644
			57	55	D1	000B4	CMPL	SCPTR, SYMSCOPE	7645	
				27	13	000B7	BEQL	13\$		
			02	14	A5	91	000B9	CMPB	20(SCPTR), #2	7647
				46	13	000BD	BEQL	15\$		
			01	14	A5	91	000BF	CMPB	20(SCPTR), #1	7651
				15	12	000C3	BNEQ	12\$		
				00000000	EF	9F	000C5	PUSHAB	P.AEI	7653
				01	DD	000CB	PUSHL	#1		
				00028362	8F	DD	000CD	PUSHL	#164706	
				03	FB	000D3	CALLS	#3, LIB\$SIGNAL		
			00	A5	D0	000DA	MOVL	16(SCPTR), SCPTR	7655	
			55	D4	11	000DE	BRB	11\$	7645	
			5A	5B	D1	000E0	CMPL	DEFDEPTH, GOOD_DEFDEPTH	7667	
				1A	12	000E3	BNEQ	14\$		

RSTACCESS
V04-000

M 3
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 237
(44)

RS
VO

FFFFFFFF	8F	59	D1	000E5	CMPL	GOOD_CAND, #-1	:	7670
		17	13	000EC	BEQL	15\$:	
		8F	BB	000EE	PUSHR	#^M<R6,R9>	:	7673
		AC	DD	000F2	PUSHL	CANDL\$:	
F4C5	CF	03	FB	000F3	CALLS	#3, CHECK_DUPLICATE	:	
	59	50	D0	000FA	MOVL	R0, GOOD_CAND	:	
		06	11	000FD	BRB	15\$:	7674
	59	56	D0	000FF	MOVL	I, GOOD_CAND	:	7681
	5A	5B	D0	00102	MOVL	DEFDEPTH, GOOD_DEFDEPTH	:	7682
FF07	56	01	08	AC	F1	00105	15\$:	7536
		50		59	D0	0010C		7691
				04	0010F	RET		7693

; Routine Size: 272 bytes, Routine Base: DBG\$CODE + 3119

```

: 7605 7694 1 ROUTINE SCOPE_RULE_NORMAL(PATHNAME, NCANDS, CANDLST, ARRAY_FLAG) =
: 7606 7695 1
: 7607 7696 1 FUNCTION
: 7608 7697 1 This routine selects the symbol from a specified list of candidate sym-
: 7609 7698 1 bols which best matches a specified pathname. This routine assumes
: 7610 7699 1 "normal" scope rules when doing so; in particular, it assumes that data
: 7611 7700 1 qualification must be complete (A.C is not accepted for A.B.C) or is not
: 7612 7701 1 present in the language. These rules suit languages like Pascal and
: 7613 7702 1 Fortran.
: 7614 7703 1
: 7615 7704 1 The list of candidate symbols is produced by DBG$STA_GETSYMBOL, and each
: 7616 7705 1 candidate is guaranteed to be in the current scope being searched. What
: 7617 7706 1 this routine must do is to determine which candidates have valid data
: 7618 7707 1 qualification, which candidate is defined at the lowest definition depth
: 7619 7708 1 (i.e., defined inner-most in the current scope), and whether that candi-
: 7620 7709 1 date is unique. The routine then returns one of three things: an indi-
: 7621 7710 1 cation that no symbol was valid, an indication that the symbol is not
: 7622 7711 1 unique, or an index pointing to the one selected candidate symbol.
: 7623 7712 1
: 7624 7713 1 INPUTS
: 7625 7714 1 PATHNAME - Pointer to the pathname descriptor for the symbol name to
: 7626 7715 1 be looked up in the symbol table.
: 7627 7716 1
: 7628 7717 1 NCANDS - The number of candidate symbols found by DBG$STA_GETSYMBOL.
: 7629 7718 1
: 7630 7719 1 CANDLST - A vector of pointers to the "candidate blocks" for the candi-
: 7631 7720 1 date symbols found by DBG$STA_GETSYMBOL. Each of these candi-
: 7632 7721 1 dates is in the scope current[ly] searched. The candidate block
: 7633 7722 1 pointers are found in CANDLST[1] through CANDLST[NCANDS].
: 7634 7723 1
: 7635 7724 1 ARRAY_FLAG - If true, the symbol we are looking up was seen in a
: 7636 7725 1 subscripted expression. This may be used to resolve
: 7637 7726 1 possible ambiguities in BASIC, where it is legal to
: 7638 7727 1 have two variables of the same name, one a scalar
: 7639 7728 1 and one an array.
: 7640 7729 1
: 7641 7730 1 OUTPUTS
: 7642 7731 1 The CANDLST index for the candidate block which best matches the path-
: 7643 7732 1 name is returned as the routine's value. If no candidate is
: 7644 7733 1 acceptable, zero is returned, and if more than one candidate
: 7645 7734 1 is acceptable (the symbol is not unique), -1 is returned.
: 7646 7735 1
: 7647 7736 1
: 7648 7737 2 BEGIN
: 7649 7738 2
: 7650 7739 2 MAP
: 7651 7740 2 PATHNAME: REF PTH$PATHNAME, ! Pointer to symbol pathname descriptor
: 7652 7741 2 CANDLST: REF VECTOR[,LONG]; ! Pointer to candidate vector
: 7653 7742 2
: 7654 7743 2 LABEL
: 7655 7744 2 CHECK_THIS_CANDIDATE; ! Label of block we want to LEAVE
: 7656 7745 2
: 7657 7746 2 LOCAL
: 7658 7747 2 CANDBLK: REF CAND_BLOCKVECTOR, ! Pointer to current "candidate block"
: 7659 7748 2 DEFDEPTH, ! Definition depth of current candidate
: 7660 7749 2 DSTPTR: REF DST$RECORD, ! Pointer to symbol DST record
: 7661 7750 2 GOOD_CAND, ! CANDLST index of best candidate so far
```

```
7662 7751 2      GOOD_DEFDEPTH,      ! Definition depth of GOOD_CAND symbol
7663 7752 2      J      ! Index for CANDBLK vector
7664 7753 2      RSTPTR: REF RST$ENTRY; ! Pointer to current symbol RST entry
7665 7754 2
7666 7755 2
7667 7756 2
7668 7757 2      ! Initially we do not have a good candidate.
7669 7758 2
7670 7759 2      GOOD_CAND = 0;
7671 7760 2      GOOD_DEFDEPTH = 1000000;
7672 7761 2
7673 7762 2
7674 7763 2      ! Loop over all the candidate blocks on the candidate list. This loop
7675 7764 2      ! searches for the best candidate symbol matching the pathname.
7676 7765 2
7677 7766 2      INCR I FROM 1 TO .NCANDS DO
7678 7767 3      BEGIN
7679 7768 3
7680 7769 3
7681 7770 3      ! Set up a labelled block to check out the current candidate. We can
7682 7771 3      ! LEAVE this block if we find that the candidate is not acceptable.
7683 7772 3
7684 7773 3      CHECK_THIS_CANDIDATE:
7685 7774 4      BEGIN
7686 7775 4      CANDBLK = .CANDLST[I];
7687 7776 4
7688 7777 4
7689 7778 4      ! Loop over the candidate's up-scope chain--that is what the CANDBLK
7690 7779 4      ! vector gives us. Reject any candidate whose data qualification in
7691 7780 4      ! the up-scope chain does not agree with that in the pathname.
7692 7781 4
7693 7782 4      J = 0;
7694 7783 4      WHILE .CANDBLK[J, CAND_RSTPTR] NEQ 0 DO
7695 7784 5      BEGIN
7696 7785 5      RSTPTR = .CANDBLK[J, CAND_RSTPTR];
7697 7786 5
7698 7787 5
7699 7788 5      ! No item before a backslash and no item not explicitly given
7700 7789 5      ! in the Pathname Descriptor may be a Data Item or Type Compo-
7701 7790 5      ! nent. This ensures complete data qualification when present.
7702 7791 5
7703 7792 6      IF (.CANDBLK[J, CAND_PINDEX] LSS .PATHNAME[PTH$B_PATHCNT] OR
7704 7793 5      .CANDBLK[J, CAND_PINDEX] EQL 0) AND
7705 7794 6      (.RSTPTR[RST$B_KIND] EQL RST$K_DATA OR
7706 7795 6      .RSTPTR[RST$B_KIND] EQL RST$K_TYPCOMP)
7707 7796 5      THEN
7708 7797 5      LEAVE CHECK_THIS_CANDIDATE;
7709 7798 5
7710 7799 5
7711 7800 5      ! No item immediately before the first dot (i.e., X in A\X or
7712 7801 5      ! A\X.B) may be a Type Component and, if there are record com-
7713 7802 5      ! ponents, that item must be a Data item. Again, this ensures
7714 7803 5      ! that data qualification is complete.
7715 7804 5
7716 7805 5      IF (.CANDBLK[J, CAND_PINDEX] EQL .PATHNAME[PTH$B_PATHCNT]) AND
7717 7806 6      ((.RSTPTR[RST$B_KIND] EQL RST$K_TYPCOMP) OR
7718 7807 7      (.PATHNAME[PTH$B_TOTCNT] GTR .PATHNAME[PTH$B_PATHCNT]) AND
```

```
7719 7808 6      .RSTPTR[RST$B_KIND] NEQ RST$K_DATA))
7720 7809 5      THEN
7721 7810 5          LEAVE CHECK_THIS_CANDIDATE;
7722 7811 5
7723 7812 5      ! If this item is part of the data qualification, it must be a
7724 7813 5      ! Type Component.
7725 7814 5
7726 7815 5      IF (.CANDBLK[J, CAND_PINDEX] GTR .PATHNAME[PTH$B_PATHCNT]) AND
7727 7816 5      (.RSTPTR[RST$B_KIND] NEQ RST$K_TYPCOMP)
7728 7817 6      THEN
7729 7818 5          LEAVE CHECK_THIS_CANDIDATE;
7730 7819 5
7731 7820 5
7732 7821 5      ! Increment the CANDBLK index and loop up-scope.
7733 7822 5
7734 7823 5      J = .J + 1;
7735 7824 5      END;
7736 7825 4
7737 7826 4
7738 7827 4
7739 7828 4      ! Pick up the definition depth from the last CANDBLK cell. Reject
7740 7829 4      ! this candidate if we already have a candidate with a smaller def-
7741 7830 4      ! inition depth (i.e., defined closer to the current scope).
7742 7831 4
7743 7832 4      DEFDEPTH = .CANDBLK[J, CAND_PINDEX];
7744 7833 4      IF .DEFDEPTH GTR .GOOD_DEFDEPTH THEN LEAVE CHECK_THIS_CANDIDATE;
7745 7834 4
7746 7835 4
7747 7836 4      ! We have a good candidate here. If we already have another candi-
7748 7837 4      ! date at the same definition depth, the symbol maybe is not unique.
7749 7838 4      ! We call a routine which attempts to resolve the ambiguity. If it
7750 7839 4      ! resolves the ambiguity, then it returns the appropriate index.
7751 7840 4      ! It returns -1 if the reference really is ambiguous.
7752 7841 4
7753 7842 4      IF .DEFDEPTH EQL .GOOD_DEFDEPTH
7754 7843 4      THEN
7755 7844 5          BEGIN
7756 7845 5              IF .GOOD_CAND EQL -1
7757 7846 5              THEN
7758 7847 5                  LEAVE CHECK_THIS_CANDIDATE;
7759 7848 5                  GOOD_CAND = CHECK_DUPLICATE(.CANDLIST, .I,
7760 7849 5                  .GOOD_CAND, .ARRAY_FLAG);
7761 7850 5                  LEAVE CHECK_THIS_CANDIDATE;
7762 7851 4              END;
7763 7852 4
7764 7853 4
7765 7854 4      ! We have a good candidate which is unique (so far) at this defini-
7766 7855 4      ! tion depth. Set GOOD_CAND accordingly.
7767 7856 4
7768 7857 4      GOOD_CAND = .I;
7769 7858 4      GOOD_DEFDEPTH = .DEFDEPTH;
7770 7859 4
7771 7860 3      END;      ! End of the CHECK_THIS_CANDIDATE block
7772 7861 3
7773 7862 2      END;      ! End of INCR loop over candidate list
7774 7863 2
7775 7864 2
```

```
: 7776      7865 2      | Return the GOOD_CAND value. This may be -1, 0, or a true CANDLST index.
: 7777      7866 2      |
: 7778      7867 2      | RETURN .GOOD_CAND;
: 7779      7868 2      |
: 7780      7869 1      | END;
```

01FC 00000 SCOPE_RULE NORMAL:

			50	D4	00002	WORD	Save R2,R3,R4,R5,R6,R7,R8	7694
			8F	D0	00004	CLRL	GOOD_CAND	7759
56	000F4240		54	D4	00008	MOVL	#1000000, GOOD_DEFDEPTH	7760
			0093	31	0000D	CLRL	1	7792
55	0C	BC44	D0	00010	1\$:	BRW	9\$	7775
		52	D4	00015		MOVL	@CANDLST[I], CANDBLK	7782
		6542	7F	00017	2\$:	CLRL	J	7783
		9E	D5	0001A		PUSHAQ	(CANDBLK)[J]	
		57	13	0001C		TSTL	@(SP)+	
		6542	7F	0001E		BEQL	7\$	
53		9E	D0	00021		PUSHAQ	(CANDBLK)[J]	7785
	04	A542	7F	00024		MOVL	@(SP)+, RSTPTR	
51		9E	D0	00028		PUSHAQ	4(CANDBLK)[J]	7792
08		08	ED	0002B		MOVL	@(SP)+, R1	
		04	14	00031		CMPZV	#8, #8, @PATHNAME, R1	
		51	D5	00033		BGTR	3\$	
		0C	12	00035		TSTL	R1	7793
06	14	A3	91	00037	3\$:	BNEQ	4\$	
		66	13	00038		CMPB	20(RSTPTR), #6	7794
0A	14	A3	91	0003D		BEQL	9\$	
		60	13	00041		CMPB	20(RSTPTR), #10	7795
51	04	BC	08	ED	00043	4\$:	BEQL	9\$
		18	12	00049		CMPZV	#8, #8, @PATHNAME, R1	7805
0A	14	A3	91	0004B		BNEQ	5\$	
		52	13	0004F		CMPB	20(RSTPTR), #10	7806
58	04	BC	9A	00051		BEQL	9\$	
08		08	ED	00055		MOVZBL	@PATHNAME, R8	7807
		06	18	0005B		CMPZV	#8, #8, @PATHNAME, R8	
06	14	A3	91	0005D		BGEQ	5\$	
		40	12	00061		CMPB	20(RSTPTR), #6	7808
51	04	BC	08	ED	00063	5\$:	BNEQ	9\$
		06	18	00069		CMPZV	#8, #8, @PATHNAME, R1	7816
0A	14	A3	91	0006B		BGEQ	6\$	
		32	12	0006F		CMPB	20(RSTPTR), #10	7817
		52	D6	00071	6\$:	BNEQ	9\$	
		A2	11	00073		INCL	J	7824
	04	A542	7F	00075	7\$:	BRB	2\$	7783
57		9E	D0	00079		PUSHAQ	4(CANDBLK)[J]	7832
56		57	D1	0007C		MOVL	@(SP)+, DEFDEPTH	
		22	14	0007F		CMPL	DEFDEPTH, GOOD_DEFDEPTH	7833
		1A	12	00081		BGTR	9\$	
FFFFFFF	8F	50	D1	00083		BNEQ	8\$	7842
		17	13	0008A		CMPL	GOOD_CAND, #-1	7845
		10	AC	DD	0008C	BEQL	9\$	
		50	DD	0008F		PUSHL	ARRAY_FLAG	7849
						PUSHL	GOOD_CAND	

RSTACCESS
V04-000

E 4
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 242
(45)

RST
V04

				54	DD	00091		PUSHL	I		7848
			OC	AC	DD	00093		PUSHL	CANDLST		
	F414	CF		04	FB	00096		CALLS	#4, CHECK_DUPLICATE		
				06	11	00098		BRB	98		7850
		50		54	DO	0009D	88:	MOVL	I, GOOD CAND		7857
		56		57	DO	000A0		MOVL	DEFDEPTH, GOOD_DEFDEPTH		7858
FF66		01		08	AC	F1 000A3	98	ACBL	NCANDS, #1, I, -18		7766
					04	000AA		RET			7869

; Routine Size: 171 bytes, Routine Base: DBG\$CODE + 3229


```
7782 7870 1 ROUTINE SCOPE_RULE_PL1(PATHNAME, NCANDS, CANDLST) =
7783 7871 1
7784 7872 1 FUNCTION
7785 7873 1     This routine selects the symbol from a specified list of candidate sym-
7786 7874 1     bols which best matches a specified pathname. This routine assumes
7787 7875 1     PL/I scope rules when doing so. This means that incomplete data quali-
7788 7876 1     fication is accepted, and that uniqueness is determined by these rules:
7789 7877 1
7790 7878 1     (1) By definition, the "lowest definition depth" is the
7791 7879 1         inner-most definition depth in the current scope at
7792 7880 1         which at least one candidate symbol is declared.
7793 7881 1
7794 7882 1     (2) If only one candidate symbol is defined at the lowest
7795 7883 1         definition depth, then that is the unique symbol we
7796 7884 1         want.
7797 7885 1
7798 7886 1     (3) If more than one symbol is defined at the lowest defi-
7799 7887 1         nition depth, but only one of them has complete data
7800 7888 1         qualification, then that is the unique symbol we want.
7801 7889 1
7802 7890 1     (4) Otherwise, the symbol is not unique.
7803 7891 1
7804 7892 1     The list of candidate symbols is produced by DBG$STA_GETSYMBOL, and each
7805 7893 1     candidate is guaranteed to be in the current scope being searched. What
7806 7894 1     this routine must do is to determine which candidates have valid data
7807 7895 1     qualification, which candidate is defined at the lowest definition depth
7808 7896 1     (i.e., defined inner-most in the current scope), and whether that candi-
7809 7897 1     date is unique. The routine then returns one of three things: an indi-
7810 7898 1     cation that no symbol was valid, an indication that the symbol is not
7811 7899 1     unique, or an index pointing to the one selected candidate symbol.
7812 7900 1
7813 7901 1 INPUTS
7814 7902 1     PATHNAME - Pointer to the pathname descriptor for the symbol name to
7815 7903 1                 be looked up in the symbol table.
7816 7904 1
7817 7905 1     NCANDS - The number of candidate symbols found by DBG$STA_GETSYMBOL.
7818 7906 1
7819 7907 1     CANDLST - A vector of pointers to the "candidate blocks" for the candi-
7820 7908 1                 date symbols found by DBG$STA_GETSYMBOL. Each of these candi-
7821 7909 1                 dates is in the scope currently searched. The candidate block
7822 7910 1                 pointers are found in CANDLST[1] through CANDLST[NCANDS].
7823 7911 1
7824 7912 1 OUTPUTS
7825 7913 1     The CANDLST index for the candidate block which best matches the path-
7826 7914 1     name is returned as the routine's value. If no candidate is
7827 7915 1     acceptable, zero is returned, and if more than one candidate
7828 7916 1     is acceptable (the symbol is not unique), -1 is returned.
7829 7917 1
7830 7918 1 BEGIN
7831 7919 2
7832 7920 2 MAP
7833 7921 2     PATHNAME: REF PTH$PATHNAME,      ! Pointer to symbol pathname descriptor
7834 7922 2     CANDLST: REF V$CTOR[.LONG];      ! Pointer to candidate vector
7835 7923 2
7836 7924 2 LABEL
7837 7925 2     CHECK_THIS_CANDIDATE;            ! Label of block we want to LEAVE
7838 7926 2
```

```

7839 7927 2
7840 7928 2
7841 7929 2
7842 7930 2
7843 7931 2
7844 7932 2
7845 7933 2
7846 7934 2
7847 7935 2
7848 7936 2
7849 7937 2
7850 7938 2
7851 7939 2
7852 7940 2
7853 7941 2
7854 7942 2
7855 7943 2
7856 7944 2
7857 7945 2
7858 7946 2
7859 7947 2
7860 7948 2
7861 7949 2
7862 7950 2
7863 7951 2
7864 7952 2
7865 7953 2
7866 7954 2
7867 7955 2
7868 7956 3
7869 7957 3
7870 7958 3
7871 7959 3
7872 7960 3
7873 7961 3
7874 7962 3
7875 7963 4
7876 7964 4
7877 7965 4
7878 7966 4
7879 7967 4
7880 7968 4
7881 7969 4
7882 7970 4
7883 7971 4
7884 7972 4
7885 7973 4
7886 7974 4
7887 7975 5
7888 7976 5
7889 7977 5
7890 7978 5
7891 7979 5
7892 7980 5
7893 7981 5
7894 7982 5
7895 7983 6

LOCAL
  CANDBLK: REF CAND_BLOCKVECTOR,  ! Pointer to current 'candidate block'
  COMPLETE_FLAG,                  ! Set to TRUE if current candidate's
                                  ! data qualification is complete
  DATAQUAL_FLAG,                 ! Set to TRUE when we are in the data
                                  ! qualification part of a name
  DEFDEPTH,                       ! Definition depth of current candidate
  DSTPTR: REF DST$RECORD,         ! Pointer to symbol DST record
  GOOD_CAND,                      ! CANDLIST index of best candidate so far
  GOOD_COMPLETE_FLAG,             ! Complete-data-qualification flag for
                                  ! the GOOD_CAND symbol
  GOOD_DEFDEPTH,                 ! Definition depth of GOOD_CAND symbol
  J,                              ! Index for CANDBLK vector
  RSTPTR: REF RST$ENTRY;          ! Pointer to current symbol RST entry

! Initially we do not have a good candidate.
GOOD_CAND = 0;
GOOD_DEFDEPTH = 1000000;
GOOD_COMPLETE_FLAG = FALSE;

! Loop over all the candidate blocks on the candidate list. This loop
! searches for the best candidate symbol matching the pathname.
INCR I FROM 1 TO .NCANDS DO
  BEGIN

    ! Set up a labelled block to check out the current candidate. We can
    ! LEAVE this block if we find that the candidate is not acceptable.
    CHECK_THIS_CANDIDATE:
    BEGIN
      CANDBLK = .CANDLIST[I];
      COMPLETE_FLAG = TRUE;

      ! Loop over the candidate's up-scope chain--that is what the CANDBLK
      ! vector gives us. Reject any candidate whose data qualification in
      ! the up-scope chain does not agree with that in the pathname.
      DATAQUAL_FLAG = TRUE;
      J = 0;
      WHILE .CANDBLK[J, CAND_RSTPTR] NEQ 0 DO
        BEGIN
          RSTPTR = .CANDBLK[J, CAND_RSTPTR];

          ! Clear DATAQUAL_FLAG if we have left the data qualification
          ! part of the name.
          IF (.CANDBLK[J, CAND_PINDEX] LSS .PATHNAME[PTH$B_PATHCNT]) AND
            (.CANDBLK[J, CAND_PINDEX] NEQ 0)
```

7896 7984 5
7897 7985 5
7898 7986 5
7899 7987 5
7900 7988 5
7901 7989 5
7902 7990 5
7903 7991 5
7904 7992 6
7905 7993 6
7906 7994 5
7907 7995 5
7908 7996 5
7909 7997 5
7910 7998 5
7911 7999 5
7912 8000 5
7913 8001 5
7914 8002 5
7915 8003 5
7916 8004 6
7917 8005 5
7918 8006 5
7919 8007 5
7920 8008 5
7921 8009 5
7922 8010 5
7923 8011 5
7924 8012 6
7925 8013 5
7926 8014 5
7927 8015 5
7928 8016 5
7929 8017 5
7930 8018 5
7931 8019 5
7932 8020 5
7933 8021 5
7934 8022 6
7935 8023 5
7936 8024 5
7937 8025 5
7938 8026 5
7939 8027 5
7940 8028 5
7941 8029 5
7942 8030 4
7943 8031 4
7944 8032 4
7945 8033 4
7946 8034 4
7947 8035 4
7948 8036 4
7949 8037 4
7950 8038 4
7951 8039 4
7952 8040 4

```
THEN
    DATAQUAL_FLAG = FALSE;

! After we leave the data qualification part going up-scope, we
! do not accept Data Items or Type Components in the name.
IF (NOT .DATAQUAL_FLAG) AND
    (.RSTPTR[RST$B_KIND] EQL RST$K_DATA OR
     .RSTPTR[RST$B_KIND] EQL RST$K_TYPCOMP)
THEN
    LEAVE CHECK_THIS_CANDIDATE;

! The last thing before the dot when there are things after the
! dot must be a Data Item or Type Component.
IF (.CANDBLK[J, CAND_PINDEX] EQL .PATHNAME[PTH$B_PATHCNT]) AND
    (.PATHNAME[PTH$B_PATHCNT] GTR .PATHNAME[PTH$B_PATHCNT]) AND
    (.RSTPTR[RST$B_KIND] NEQ RST$K_DATA) AND
    (.RSTPTR[RST$B_KIND] NEQ RST$K_TYPCOMP)
THEN
    LEAVE CHECK_THIS_CANDIDATE;

! After the dot, everything must be Type Components.
IF (.CANDBLK[J, CAND_PINDEX] GTR .PATHNAME[PTH$B_PATHCNT]) AND
    (.RSTPTR[RST$B_KIND] NEQ RST$K_TYPCOMP)
THEN
    LEAVE CHECK_THIS_CANDIDATE;

! If we are in the data qualification part and PINDEX is zero,
! we have an RST entry on the up-scope chain whose name is not
! given in the Pathname Descriptor. This means that data quali-
! fication is not complete for this variable.
IF .DATAQUAL_FLAG AND (.CANDBLK[J, CAND_PINDEX] EQL 0)
THEN
    COMPLETE_FLAG = FALSE;

! Increment the CANDBLK index and loop up-scope.
J = J + 1;
END;

! Pick up the definition depth from the last CANDBLK cell. Reject
! this candidate if we already have a candidate with a smaller def-
! inition depth (i.e., defined closer to the current scope).
DEFDEPTH = .CANDBLK[J, CAND_PINDEX];
IF .DEFDEPTH GTR .GOOD_DEFDEPTH THEN LEAVE CHECK_THIS_CANDIDATE;
```

```
: 7953      8041  4      ! We have a good candidate here.  If we already have another candi-
: 7954      8042  4      ! date at the same definition depth, the symbol maybe is not unique.
: 7955      8043  4      ! If only one of the two candidates has complete data qualification,
: 7956      8044  4      ! we accept that one candidate as being the one we want (so far).
: 7957      8045  4      ! Otherwise, we call a routine which attempts to resolve the ambiguity.  If it
: 7958      8046  4      ! resolves the ambiguity, then it returns the appropriate index.
: 7959      8047  4      ! It returns -1 if the reference really is amiguous.
: 7960      8048  4
: 7961      8049  4      IF (.DEFDEPTH EQL .GOOD_DEFDEPTH) AND
: 7962      8050  5      (.GOOD_COMPLETE_FLAG OR NOT .COMPLETE_FLAG)
: 7963      8051  4      THEN
: 7964      8052  5      BEGIN
: 7965      8053  6      IF (.COMPLETE_FLAG OR NOT .GOOD_COMPLETE_FLAG)
: 7966      8054  5      THEN
: 7967      8055  6      BEGIN
: 7968      8056  6      IF .GOOD_CAND EQL -1
: 7969      8057  6      THEN
: 7970      8058  6      LEAVE CHECK_THIS_CANDIDATE;
: 7971      8059  6      GOOD_CAND = CHECK_DUPLICATE(.CANDLIST, .I, .GOOD_CAND);
: 7972      8060  6      IF .GOOD_CAND EQL .I
: 7973      8061  6      THEN
: 7974      8062  6      GOOD_COMPLETE_FLAG = .COMPLETE_FLAG;
: 7975      8063  5      END;
: 7976      8064  5      LEAVE CHECK_THIS_CANDIDATE;
: 7977      8065  4      END;
: 7978      8066  4
: 7979      8067  4
: 7980      8068  4      ! We have a good candidate which is unique (so far) at this defini-
: 7981      8069  4      ! tion depth.  Set GOOD_CAND accordingly.
: 7982      8070  4
: 7983      8071  4      GOOD_CAND = .I;
: 7984      8072  4      GOOD_DEFDEPTH = .DEFDEPTH;
: 7985      8073  4      GOOD_COMPLETE_FLAG = .COMPLETE_FLAG;
: 7986      8074  4
: 7987      8075  3      END;
: 7988      8076  3      ! End of the CHECK_THIS_CANDIDATE block
: 7989      8077  2      END;
: 7990      8078  2      ! End of INCR loop over candidate list
: 7991      8079  2
: 7992      8080  2      ! Return the GOOD_CAND value.  This may be -1, 0, or a true CANDLIST index.
: 7993      8081  2      !
: 7994      8082  2      RETURN .GOOD_CAND;
: 7995      8083  2
: 7996      8084  1      END;
```

```
OFFC 00000 SCOPE_RULE PL1:
                                WORD      Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      : 7870
                                CLRL      GOOD_CAND                               : 7947
58 000F4240      50  D4 00002      MOVL      #1000000, GOOD_DEFDEPTH              : 7948
                                57  D4 0000B      CLRL      GOOD_COMPLETE_FLAG      : 7949
                                54  D4 0000D      CLRL      I                       : 7982
                                00B8 31 0000F 1$:  BRW      14$                     :
55      0C BC44  D0 00012 2$:  MOVL      @CANDLIST[I], CANDBLK                    : 7964
```

56			01	DO	00017	MOVL	#1, COMPLETE_FLAG	7965	
5A			01	DO	0001A	MOVL	#1, DATAQUAL_FLAG	7972	
			52	D4	0001D	CLRL	J	7973	
			6542	7F	0001F	PUSHAQ	(CANDBLK)[J]	7974	
			9E	D5	00022	TSTL	@(SP)+		
			65	13	00024	BEQL	9\$		
			6542	7F	00026	PUSHAQ	(CANDBLK)[J]	7976	
53			9E	DO	00029	MOVL	@(SP)+, RSTPTR		
			04 A542	7F	0002C	PUSHAQ	4(CANDBLK)[J]	7982	
51			9E	DO	00030	MOVL	@(SP)+, R1		
08			08	ED	00033	CMPZV	#8, #8, @PATHNAME, R1		
			06	15	00039	BLEQ	4\$		
			51	D5	0003B	TSTL	R1	7983	
			02	13	0003D	BEQL	4\$		
			5A	D4	0003F	CLRL	DATAQUAL_FLAG	7985	
0C			5A	E8	00041	BLBS	DATAQUAL_FLAG, 5\$	7991	
03			14 A3	91	00044	CMPB	20(RSTPTR), #6	7992	
			C5	13	00048	BEQL	1\$		
0A			14 A3	91	0004A	CMPB	20(RSTPTR), #10	7993	
			7A	13	0004E	BEQL	14\$		
51	04	BC	08	08	ED	00050	CMPZV	#8, #8, @PATHNAME, R1	8001
			18	12	00056	BNEQ	6\$		
5B	04	BC	5B	04 BC	9A	00058	MOVZBL	@PATHNAME, R11	8002
			08	08	ED	0005C	CMPZV	#8, #8, @PATHNAME, R11	
			0C	18	00062	BGEQ	6\$		
			06	14 A3	91	00064	CMPB	20(RSTPTR), #6	8003
				06	13	00068	BEQL	6\$	
			0A	14 A3	91	0006A	CMPB	20(RSTPTR), #10	8004
				5A	12	0006E	BNEQ	14\$	
51	04	BC	08	08	ED	00070	CMPZV	#8, #8, @PATHNAME, R1	8011
			0A	14 A3	91	00078	CMPB	20(RSTPTR), #10	8012
				4C	12	0007C	BNEQ	14\$	
			06	5A	E9	0007E	7\$: BLBC	DATAQUAL_FLAG, 8\$	8022
				51	D5	00081	TSTL	R1	
				02	12	00083	BNEQ	8\$	
				56	D4	00085	CLRL	COMPLETE_FLAG	8024
				52	D6	00087	8\$: INCL	J	8029
				94	11	00089	BRB	3\$	7974
			04 A542	7F	0008B	9\$: PUSHAQ	4(CANDBLK)[J]	8037	
59			9E	DO	0008F	MOVL	@(SP)+, DEFDEPTH		
58			59	D1	00092	CMPL	DEFDEPTH, GOOD_DEFDEPTH	8038	
			33	14	00095	BGTR	14\$		
			28	12	00097	BNEQ	12\$	8049	
03			57	E8	00099	BLBS	GOOD_COMPLETE_FLAG, 10\$	8050	
22			56	E8	0009C	BLBS	COMPLETE_FLAG, 12\$		
03			56	E8	0009F	10\$: BLBS	COMPLETE_FLAG, 11\$	8053	
25			57	E8	000A2	BLBS	GOOD_COMPLETE_FLAG, 14\$		
FFFFFFFFFF			50	D1	000A5	11\$: CMPL	GOOD_CAND, #-T	8056	
			1C	13	000AC	BEQL	14\$		
			50	DD	000AE	PUSHL	GOOD_CAND	8059	
			54	DD	000B0	PUSHL	I		
F34A			0C AC	DD	000B2	PUSHL	CANDLST		
			CF	03	FB	000B5	CALLS	#3, CHECK_DUPLICATE	
54			50	D1	000BA	CMPL	GOOD_CAND, I	8060	
			08	12	000BD	BNEQ	14\$		
			06	11	000BF	BRB	13\$	8062	

RSTACCESS
V04-000

K 4
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACCESS.B32;1

Page 248
(46)

RS
VO

		50		54	D0 000C1 12\$:	MOVL	I, GOOD CAND	:	8071
		58		59	D0 000C4	MOVL	DEFDEPTH, GOOD_DEFDEPTH	:	8072
		57		56	D0 000C7 13\$:	MOVL	COMPLETE_FLAG, -GOOD_COMPLETE_FLAG	:	8073
FF41	54	01	08	AC	F1 000CA 14\$:	ACBL	NCANDS, #1, I, 2\$:	7955
					04 000D1	RET		:	8084

; Routine Size: 210 bytes, Routine Base: DBG\$CODE + 32D4

```

7998 8085 1 ROUTINE SETCONTEXT_ERROR_HANDLER (SIGARG, MECHARG, ENBLARG) =
7999 8086 1
8000 8087 1 FUNCTION
8001 8088 1     This routine is the error handler for the DBG$STA SETCONTEXT routine.
8002 8089 1     It handles Access Violations which occur during the following of stack
8003 8090 1     call frames. Since such access violations are not normally caused by
8004 8091 1     errors in Debug but rather by errors in the user program (e.g., by
8005 8092 1     clobbered FP), we give a special message for this kind of access
8006 8093 1     violation.
8007 8094 1     The message says that there is a bad frame pointer or call frame in
8008 8095 1     the stack.
8009 8096 1
8010 8097 1 INPUTS
8011 8098 1     SIGARG - The signal argument vector.
8012 8099 1
8013 8100 1     MECHARG - The mechanism argument vector.
8014 8101 1
8015 8102 1     ENBLARG - The enable argument vector (not used here).
8016 8103 1
8017 8104 1 OUTPUTS
8018 8105 1     For the SS$_ACCVIO error, the DBG$_BADFRAME informational message is
8019 8106 1     signaled, and the stack is unwound to leave DBG$STA_SETCONTEXT.
8020 8107 1     For all other errors, this routine just resignals.
8021 8108 1
8022 8109 1
8023 8110 2 BEGIN
8024 8111 2
8025 8112 2 MAP
8026 8113 2     SIGARG: REF VECTOR[,LONG];           ! Pointer to the signal argument vector
8027 8114 2
8028 8115 2
8029 8116 2     ! If this is anything other than an access violation, just resignal it.
8030 8117 2
8031 8118 2     IF .SIGARG[1] NEQ SS$_ACCVIO THEN RETURN SS$_RESIGNAL;
8032 8119 2
8033 8120 2
8034 8121 2     ! It is an access violation. Signal the informational and unwind.
8035 8122 2
8036 8123 2     SIGNAL (DBG$_BADFRAME);
8037 8124 2     SETUNWIND ();
8038 8125 2     RETURN 0;
8039 8126 2
8040 8127 1 END;
```

```

                                0000 00000 SETCONTEXT_ERROR_HANDLER:
                                .WORD    Save nothing
                                50        04    AC    D0 00002    MOVL    SIGARG, R0
                                0C        04    A0    D1 00006    CMPL    4(R0), #12
                                06        13 0000A    BEQL    1$
                                50        0918 8F    3C 0000C    MOVZWL  #2328, R0
                                04        00011    RET
                                00000000G 00 00028693 8F    DD 00012 1$:    PUSHL  #165523
                                01        FB 00018    CALLS  #1, LIB$SIGNAL
```

8085
8118

8123

```

M 4
16-Sep-1984 02:48:17 VAX-11 Bliss-32 v4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACKACCESS.B32;1

```

Page 250
(47)

```

7E 7C 0001F
02 FB 00021
50 D4 00028
    04 0002A

```

```

CLRQ      -(SP)
CALLS     #2, SYSSUNWIND
CLRL      R0
RET

```

: 8124
 : 8125
 : 8127

```
; Routine Size: 43 bytes,   Routine Base: DBG$CODE + 33A6
```

[illegible]


```
8042 8128 1 ROUTINE STACK_MACHINE(STK_CODE_PTR, RESULT_PTR, FRAMEPTR): NOVALUE =
8043 8129 1
8044 8130 1 FUNCTION
8045 8131 1 This routine evaluates "Stack Machine" code from a Materialization Spec
8046 8132 1 in a DST Value Spec. It accepts as input a pointer to the Stack Machine
8047 8133 1 "routine" (i.e., the "code") to be evaluated. That "routine" is then
8048 8134 1 evaluated on a stack built in a temporary memory block. Upon return,
8049 8135 1 the address of the computed value in the temporary memory block is
8050 8136 1 returned as the result of the evaluation.
8051 8137 1
8052 8138 1 INPUTS
8053 8139 1 STK_CODE_PTR - The address of the first byte of "Stack Machine" code.
8054 8140 1 Evaluation of this "code" starts at this address and con-
8055 8141 1 tinues until the DST&K_STK_STOP command is reached.
8056 8142 1
8057 8143 1 RESULT_PTR - The address of a longword location to receive the result
8058 8144 1 pointer.
8059 8145 1
8060 8146 1 FRAMEPTR - The address of a longword location to receive the frame
8061 8147 1 pointer associated with the result location.
8062 8148 1
8063 8149 1 OUTPUTS
8064 8150 1 RESULT_PTR - A pointer to the result of evaluating the stack machine
8065 8151 1 routine is returned to RESULT_PTR.
8066 8152 1
8067 8153 1 FRAMEPTR - The Frame Pointer (FP) value of the register set used in
8068 8154 1 the stack machine computations is returned to FRAMEPTR if
8069 8155 1 any register was used in the computations. If no register
8070 8156 1 value was used, zero is returned to FRAMEPTR.
8071 8157 1
8072 8158 1 No value is returned by routine STACK_MACHINE.
8073 8159 1
8074 8160 1
8075 8161 2 BEGIN
8076 8162 2
8077 8163 2 MAP
8078 8164 2 RESULT_PTR: REF VECTOR[1], ! Pointer to result location
8079 8165 2 FRAMEPTR: REF VECTOR[1]; ! Pointer to frame pointer location
8080 8166 2
8081 8167 2 LITERAL
8082 8168 2 STACK_SIZE = 256;
8083 8169 2
8084 8170 2 LOCAL
8085 8171 2 CALL RESULT, ! Result of embedded routine call
8086 8172 2 STACK_PTR, ! Pointer to the top of stack.
8087 8173 2 OVERFLOW_POINT, ! Pointer to stack upper limit.
8088 8174 2 UNDERFLOW_POINT, ! Pointer to stack lower limit.
8089 8175 2 INSTRUC : REF VECTOR [,BYTE]; ! Pointer to the current stack instruct
8090 8176 2
8091 8177 2 MACRO
8092 8178 2 TOP_CELL = (.STACK_PTR) %,
8093 8179 2 SECOND_CELL = (.STACK_PTR + 4) %,
8094 8180 2 PUSH(1) = STACK_PTR = .STACK_PTR - (1)*XUPVAL;
8095 M 8181 2 IF .STACK_PTR LESS .OVERFLOW_POINT
8096 M 8182 2 THEN
8097 8183 2 $DBG_ERROR('RSTACCESS\STACK_MACHINE 10') %,
8098 8184 2
```

```
: 8099 M 8185 2
: 8100 M 8186 2
: 8101 M 8187 2
: 8102 M 8188 2
: 8103 M 8189 2
: 8104 M 8190 2
: 8105 M 8191 2
: 8106 M 8192 2
: 8107 M 8193 2
: 8108 M 8194 2
: 8109 M 8195 2
: 8110 M 8196 2
: 8111 M 8197 2
: 8112 M 8198 2
: 8113 M 8199 2
: 8114 M 8200 2
: 8115 M 8201 2
: 8116 M 8202 2
: 8117 M 8203 2
: 8118 M 8204 2
: 8119 M 8205 2
: 8120 M 8206 2
: 8121 M 8207 2
: 8122 M 8208 2
: 8123 M 8209 2
: 8124 M 8210 2
: 8125 M 8211 2
: 8126 M 8212 2
: 8127 M 8213 2
: 8128 M 8214 2
: 8129 M 8215 2
: 8130 M 8216 2
: 8131 M 8217 2
: 8132 M 8218 2
: 8133 M 8219 2
: 8134 M 8220 2
: 8135 M 8221 2
: 8136 M 8222 3
: 8137 M 8223 3
: 8138 M 8224 3
: 8139 M 8225 3
: 8140 M 8226 3
: 8141 M 8227 3
: 8142 M 8228 3
: 8143 M 8229 3
: 8144 M 8230 3
: 8145 M 8231 3
: 8146 M 8232 3
: 8147 M 8233 3
: 8148 M 8234 3
: 8149 M 8235 3
: 8150 M 8236 3
: 8151 M 8237 3
: 8152 M 8238 3
: 8153 M 8239 3
: 8154 M 8240 3
: 8155 M 8241 3
```

```
POP(I) = STACK_PTR = .STACK_PTR + (I)*%UPVAL;
IF .STACK_PTR GTRA .UNDERFLOW_POINT
THEN
    $DBG_ERROR('RSTACCESS\STACK_MACHINE 20') %;

PUSH_BYTE(I) = STACK_PTR =
    .STACK_PTR -
    ((I) + (IF (I) MOD %UPVAL NEQ 0
    THEN %UPVAL - ((I) MOD %UPVAL)
    ELSE 0));
IF .STACK_PTR LSSA .OVERFLOW_POINT
THEN
    $DBG_ERROR('RSTACCESS\STACK_MACHINE 30') %;

CHECK_CELLS(I) = IF .UNDERFLOW_POINT - .STACK_PTR LSS (I)*%UPVAL
THEN
    $DBG_ERROR('RSTACCESS\STACK_MACHINE 40') %;

! Initialize the stack and the pointer to the instruction stream.
!
OVERFLOW_POINT = DBG$GET_TEMP_MEM(STACK_SIZE);
UNDERFLOW_POINT = .OVERFLOW_POINT + 4*STACK_SIZE;
STACK_PTR = .UNDERFLOW_POINT;
INSTRUC = .STK_CODE_PTR;

! Initialize FRAMEPTR to zero--this will be changed if registers are used.
!
FRAMEPTR[0] = 0;

! Evaluate the Stack Machine "routine" by looping through its instructions
! until we reach the Stop command.
!
WHILE .INSTRUC[0] NEQ DST$K_STK_STOP DO
    BEGIN

        ! Do a CASE on the current Stack Machine Op Code and execute each
        ! op code as appropriate.
        !
        CASE .INSTRUC[0] FROM DST$K_STK_LOW TO DST$K_STK_HIGH OF
            SET

                ! Push the value of a register on the stack.
                !
                [DST$K_STK_PUSHR0,
                DST$K_STK_PUSHR1,
                DST$K_STK_PUSHR2,
                DST$K_STK_PUSHR3,
                DST$K_STK_PUSHR4,
                DST$K_STK_PUSHR5,
                DST$K_STK_PUSHR6,
                DST$K_STK_PUSHR7,
```

```
: 8156 8242 3
: 8157 8243 3
: 8158 8244 3
: 8159 8245 3
: 8160 8246 3
: 8161 8247 3
: 8162 8248 3
: 8163 8249 3
: 8164 8250 4
: 8165 8251 4
: 8166 8252 4
: 8167 8253 4
: 8168 8254 5
: 8169 8255 5
: 8170 8256 5
: 8171 8257 5
: 8172 8258 5
: 8173 8259 5
: 8174 8260 5
: 8175 8261 5
: 8176 8262 5
: 8177 8263 5
: 8178 8264 5
: 8179 8265 5
: 8180 8266 5
: 8181 8267 5
: 8182 8268 5
: 8183 8269 5
: 8184 8270 5
: 8185 8271 5
: 8186 8272 5
: 8187 8273 5
: 8188 8274 4
: 8189 8275 4
: 8190 8276 4
: 8191 8277 4
: 8192 8278 5
: 8193 8279 5
: 8194 8280 5
: 8195 8281 5
: 8196 8282 5
: 8197 8283 4
: 8198 8284 4
: 8199 8285 4
: 8200 8286 4
: 8201 8287 3
: 8202 8288 3
: 8203 8289 3
: 8204 8290 3
: 8205 8291 3
: 8206 8292 3
: 8207 8293 3
: 8208 8294 3
: 8209 8295 4
: 8210 8296 4
: 8211 8297 4
: 8212 8298 4
```

```
DST$K_STK_PUSH8,
DST$K_STK_PUSH9,
DST$K_STK_PUSH10,
DST$K_STK_PUSH11,
DST$K_STK_PUSHRAP,
DST$K_STK_PUSHRFP,
DST$K_STK_PUSHRSP,
DST$K_STK_PUSHRPC];
BEGIN
LOCAL REGISTR;
PUSH(1);
REGISTR =
(CASE .INSTRUC[0]
FROM DST$K_STK_PUSH8 TO DST$K_STK_PUSHRPC OF
SET
[DST$K_STK_PUSH8]: 0;
[DST$K_STK_PUSH9]: 1;
[DST$K_STK_PUSH10]: 2;
[DST$K_STK_PUSH11]: 3;
[DST$K_STK_PUSHRAP]: 4;
[DST$K_STK_PUSHRFP]: 5;
[DST$K_STK_PUSHRSP]: 6;
[DST$K_STK_PUSHRPC]: 7;
[DST$K_STK_PUSH8]: 8;
[DST$K_STK_PUSH9]: 9;
[DST$K_STK_PUSH10]: 10;
[DST$K_STK_PUSH11]: 11;
[DST$K_STK_PUSHRAP]: 12;
[DST$K_STK_PUSHRFP]: 13;
[DST$K_STK_PUSHRSP]: 14;
[DST$K_STK_PUSHRPC]: 15;
TES
);
IF .DBG$REG_VECTOR[.REGISTR] NEQ 0
THEN
BEGIN
TOP_CELL = .DBG$REG_VALUES[.REGISTR];
INSTRUC = .INSTRUC + 1;
END
ELSE
VALSPEC_SCOPE_ERROR();

FRAMEPTR[0] = .DBG$REG_VALUES[13];
END;

! PUSH IMMEDIATE 1, 2, or 4 bytes following this opcode
! BYTE WORD OR LONG are sign extended to 32 bits and PUSHed
! on the stack
[DST$K_STK_PUSHIMB]:
BEGIN
LOCAL OPERAND : REF VECTOR [,BYTE,SIGNED];
PUSH(1);
OPERAND = INSTRUC[1];
```

```
: 8213 8299 4
: 8214 8300 4
: 8215 8301 3
: 8216 8302 3
: 8217 8303 3
: 8218 8304 4
: 8219 8305 4
: 8220 8306 4
: 8221 8307 4
: 8222 8308 4
: 8223 8309 4
: 8224 8310 3
: 8225 8311 3
: 8226 8312 3
: 8227 8313 4
: 8228 8314 4
: 8229 8315 4
: 8230 8316 4
: 8231 8317 4
: 8232 8318 4
: 8233 8319 3
: 8234 8320 3
: 8235 8321 3
: 8236 8322 3
: 8237 8323 3
: 8238 8324 3
: 8239 8325 3
: 8240 8326 3
: 8241 8327 3
: 8242 8328 4
: 8243 8329 4
: 8244 8330 4
: 8245 8331 4
: 8246 8332 3
: 8247 8333 3
: 8248 8334 3
: 8249 8335 3
: 8250 8336 3
: 8251 8337 3
: 8252 8338 3
: 8253 8339 3
: 8254 8340 4
: 8255 8341 4
: 8256 8342 4
: 8257 8343 4
: 8258 8344 3
: 8259 8345 3
: 8260 8346 3
: 8261 8347 4
: 8262 8348 4
: 8263 8349 4
: 8264 8350 4
: 8265 8351 4
: 8266 8352 4
: 8267 8353 3
: 8268 8354 3
: 8269 8355 3
```

```
TOP_CELL = .OPERAND[0];
INSTRUC = .INSTRUC + 2;
END;

[DST$K_STK_PUSHIMW]:
BEGIN
LOCAL OPERAND : REF VECTOR [,WORD, SIGNED];
PUSH(1);
OPERAND = INSTRUC[1];
TOP_CELL = .OPERAND[0];
INSTRUC = .INSTRUC + 3;
END;

[DST$K_STK_PUSHIML]:
BEGIN
LOCAL OPERAND : REF VECTOR [,LONG];
PUSH(1);
OPERAND = INSTRUC[1];
TOP_CELL = .OPERAND[0];
INSTRUC = .INSTRUC + 5;
END;

: PUSH IMMEDIATE VARIABLE      The byte following the opcode is
:                               interpreted as an unsigned byte count.
:                               A block of data, immediately following
:                               the count byte, is PUSHed on the stack.
:

[DST$K_STK_PUSHIM_VAR]:
BEGIN
PUSH BYTE(.INSTRUC[1]);
CH$MOVE(.INSTRUC[1], INSTRUC[2], TOP_CELL );
INSTRUC = INSTRUC[2] + .INSTRUC[1];
END;

: PUSH IMMEDIATE UNSIGNED      1 or 2 bytes following this opcode
: BYTE OR WORD                 are zero extended to 32 bits and PUSHed
:                               on the stack
:

[DST$K_STK_PUSHIMBU]:
BEGIN
PUSH(1);
TOP_CELL = .INSTRUC[1];
INSTRUC = .INSTRUC + 2;
END;

[DST$K_STK_PUSHIMWU]:
BEGIN
LOCAL OPERAND : REF VECTOR [,WORD];
PUSH(1);
OPERAND = INSTRUC[1];
TOP_CELL = .OPERAND[0];
INSTRUC = .INSTRUC + 3;
END;
```

```
8270 8356 3
8271 8357 3
8272 8358 3
8273 8359 3
8274 8360 3
8275 8361 3
8276 8362 4
8277 8363 4
8278 8364 4
8279 8365 4
8280 8366 4
8281 8367 3
8282 8368 3
8283 8369 3
8284 8370 4
8285 8371 4
8286 8372 4
8287 8373 4
8288 8374 4
8289 8375 3
8290 8376 3
8291 8377 3
8292 8378 4
8293 8379 4
8294 8380 4
8295 8381 4
8296 8382 4
8297 8383 3
8298 8384 3
8299 8385 3
8300 8386 3
8301 8387 3
8302 8388 3
8303 8389 3
8304 8390 3
8305 8391 3
8306 8392 4
8307 8393 4
8308 8394 4
8309 8395 4
8310 8396 4
8311 8397 3
8312 8398 3
8313 8399 3
8314 8400 4
8315 8401 4
8316 8402 4
8317 8403 4
8318 8404 4
8319 8405 3
8320 8406 3
8321 8407 3
8322 8408 3
8323 8409 3
8324 8410 3
8325 8411 3
8326 8412 3
```

```
! PUSH INDIRECT      The top stack cell is popped and 1, 2, or 4
! BYTE WORD OR LONG bytes at the address given by the popped
!                      stack cell are sign extended to 32 bits and
!                      pushed on the stack.
[DST$K_STK_PUSHINB]:
  BEGIN
  LOCAL OPERAND : REF VECTOR [,BYTE, SIGNED];
  OPERAND = .TOP CELL;
  TOP CELL = .OPERAND[0];
  INSTRUC = .INSTRUC + 1;
  END;

[DST$K_STK_PUSHINW]:
  BEGIN
  LOCAL OPERAND : REF VECTOR [,WORD, SIGNED];
  OPERAND = .TOP CELL;
  TOP CELL = .OPERAND[0];
  INSTRUC = .INSTRUC + 1;
  END;

[DST$K_STK_PUSHINL]:
  BEGIN
  LOCAL OPERAND : REF VECTOR [,LONG];
  OPERAND = .TOP CELL;
  TOP CELL = .OPERAND[0];
  INSTRUC = .INSTRUC + 1;
  END;

! PUSH INDIRECT UNSIGNED  The top stack cell is popped and 1 or 2
! BYTE OR WORD           bytes at the address given by the popped
!                      stack cell are zero extended to 32 bits
!                      and pushed on the stack.
[DST$K_STK_PUSHINBU]:
  BEGIN
  LOCAL OPERAND : REF VECTOR [,BYTE];
  OPERAND = .TOP CELL;
  TOP CELL = .OPERAND[0];
  INSTRUC = .INSTRUC + 1;
  END;

[DST$K_STK_PUSHINWU]:
  BEGIN
  LOCAL OPERAND : REF VECTOR [,WORD];
  OPERAND = .TOP CELL;
  TOP CELL = .OPERAND[0];
  INSTRUC = .INSTRUC + 1;
  END;

! ADD                  The top two stack cells are added and
!                      replaced by a single cell containing
!                      their sum
[DST$K_STK_ADD]:
```

```

: 8327      8413 4
: 8328      8414 4
: 8329      8415 4
: 8330      8416 4
: 8331      8417 4
: 8332      8418 3
: 8333      8419 3
: 8334      8420 3
: 8335      8421 3
: 8336      8422 3
: 8337      8423 3
: 8338      8424 3
: 8339      8425 3
: 8340      8426 4
: 8341      8427 4
: 8342      8428 4
: 8343      8429 4
: 8344      8430 4
: 8345      8431 3
: 8346      8432 3
: 8347      8433 3
: 8348      8434 3
: 8349      8435 3
: 8350      8436 3
: 8351      8437 3
: 8352      8438 3
: 8353      8439 4
: 8354      8440 4
: 8355      8441 4
: 8356      8442 4
: 8357      8443 4
: 8358      8444 3
: 8359      8445 3
: 8360      8446 3
: 8361      8447 3
: 8362      8448 3
: 8363      8449 3
: 8364      8450 3
: 8365      8451 3
: 8366      8452 4
: 8367      8453 4
: 8368      8454 4
: 8369      8455 4
: 8370      8456 4
: 8371      8457 4
: 8372      8458 4
: 8373      8459 5
: 8374      8460 5
: 8375      8461 5
: 8376      8462 5
: 8377      8463 5
: 8378      8464 3
: 8379      8465 3
: 8380      8466 3
: 8381      8467 3
: 8382      8468 3
: 8383      8469 3

      BEGIN
      CHECK CELLS(2);
      SECOND_CELL = .TOP_CELL + .SECOND_CELL;
      POP(1);
      INSTRUC = .INSTRUC + 1;
      END;

: SUBTRACT      The second stack cell is subtracted from
:               the first stack cell. Both are popped.
:               Their difference is PUSHed.
:
: [DSTSK_STK_SUB]:
:   BEGIN
:   CHECK CELLS(2);
:   SECOND_CELL = .TOP_CELL - .SECOND_CELL;
:   POP(1);
:   INSTRUC = .INSTRUC + 1;
:   END;

: MULTIPLY      The top two stack cells are multiplied
:               and replaced by a single cell containing
:               their product
:
: [DSTSK_STK_MULT]:
:   BEGIN
:   CHECK CELLS(2);
:   SECOND_CELL = (.TOP_CELL)*(.SECOND_CELL);
:   POP(1);
:   INSTRUC = .INSTRUC + 1;
:   END;

: DIVIDE        The top stack cell is divided by the
:               secondstack cell. Both are popped.
:               Their goutient is PUSHed.
:
: [DSTSK_STK_DIV]:
:   BEGIN
:   CHECK CELLS(2);
:   IF (.SECOND_CELL) EQL 0
:   THEN
:       $DBG_ERROR('RSTACCESS\STACK_MACHINE 50')
:   ELSE
:       BEGIN
:       SECOND_CELL = (.TOP_CELL)/(.SECOND_CELL);
:       POP(1);
:       INSTRUC = .INSTRUC + 1;
:       END
:   END;

: LOGICAL SHIFT The top stack cell is interpreted as
:               the number of bit positions to shift the
:               second stack cell. Both are popped.
```

8384 8470 3
8385 8471 3
8386 8472 3
8387 8473 4
8388 8474 4
8389 8475 4
8390 8476 4
8391 8477 5
8392 8478 5
8393 8479 5
8394 8480 5
8395 8481 5
8396 8482 4
8397 8483 5
8398 8484 5
8399 8485 5
8400 8486 5
8401 8487 5
8402 8488 5
8403 8489 6
8404 8490 6
8405 8491 6
8406 8492 6
8407 8493 6
8408 8494 5
8409 8495 5
8410 8496 5
8411 8497 5
8412 8498 5
8413 8499 5
8414 8500 5
8415 8501 6
8416 8502 6
8417 8503 6
8418 8504 6
8419 8505 6
8420 8506 6
8421 8507 6
8422 8508 5
8423 8509 5
8424 8510 3
8425 8511 3
8426 8512 3
8427 8513 3
8428 8514 3
8429 8515 3
8430 8516 3
8431 8517 3
8432 8518 3
8433 8519 4
8434 8520 4
8435 8521 4
8436 8522 4
8437 8523 4
8438 8524 4
8439 8525 4
8440 8526 4

```

:                                     The shifted second cell is PUSHed.
[DST$K_STK_LSH]:
BEGIN
CHECK CELLS(2);
IF ABS( .TOP_CELL ) GEQ %BPVAL
THEN
BEGIN
POP(1);
TOP_CELL = 0;
INSTRUC = .INSTRUC + 1;
END
ELSE
BEGIN
IF .TOP_CELL GTR 0
THEN
: Number of bit positions is positive, shift to the left.
:
BEGIN
SECOND_CELL = (.SECOND_CELL)^(.TOP_CELL);
POP(1);
INSTRUC = .INSTRUC + 1;
END
ELSE
: Number of bit positions is negative, shift to the right.
: This is a logical, rather than an arithmetic shift, so
: we'll have to do some magic, rather than use the BLISS
: shift operator.
:
BEGIN
LOCAL POSITION, SIZ;
POSITION = -.TOP_CELL;
SIZ = %BPVAL - POSITION;
SECOND_CELL = (.SECOND_CELL)<.POSITION, .SIZ>;
POP(1);
INSTRUC = .INSTRUC + 1;
END;
END;
END;

: ROTATE                                     The top stack cell is interpreted as the
:                                     number of bit positions to rotate the
:                                     second stack cell. Both are popped.
:                                     The rotated second cell is PUSHed.
[DST$K_STK_ROT]:
BEGIN
LOCAL BITS_TO_ROT;
CHECK CELLS(2);
BITS_TO_ROT = .TOP_CELL MOD %BPVAL;
IF .BITS_TO_ROT GTR 0
THEN
: Number of bit positions is positive, rotate to the left.
```

```

BEGIN
LOCAL OPERAND, TARG_POS, SRC_POS, SIZ;
OPERAND = .(SECOND_CELL);

! Move the low order bits of the source to the high order
! bits of the target and the high order bits of the source to
! the low order bits of the target.

      31                                     0
source +-----+-----+
       |             |             |
       |             |             |
       +-----+-----+
       |             |             |
       |             |             |
       +-----+-----+
          :BITS_TO_ROT:
           |             |
           |             |
           +-----+-----+
              \ /         \ /
target   +-----+-----+
        |             |             |
        +-----+-----+

TARG_POS = .BITS_TO_ROT;
SRC_POS = 0;
SIZ = %BPVAL - TARG_POS;
(SECOND_CELL)<.TARG_POS, .SIZ> = .OPERAND<.SRC_POS, .SIZ>;

! Move the high order bits of the source to the low order
! bits of the target.

TARG_POS = 0;
SRC_POS = %BPVAL - .BITS_TO_ROT;
SIZ = .BITS_TO_ROT;
(SECOND_CELL)<.TARG_POS, .SIZ> = .OPERAND<.SRC_POS, .SIZ>;

! Adjust the stack pointer.
POP(1);
INSTRUC = .INSTRUC + 1;
END

ELSE

! Number of bit positions is negative, rotate to the right.

BEGIN
LOCAL OPERAND, TARG_POS, SRC_POS, SIZ;
OPERAND = .SECOND_CELL;

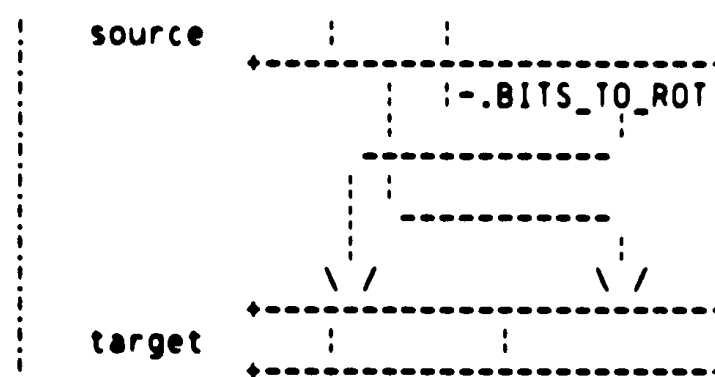
! Move the high order bits of the source to the low order
! bits of the target and the low order bits of the source to
! the high order bits of the target.

      31                                     0

```



```
8498 8584 5
8499 8585 5
8500 8586 5
8501 8587 5
8502 8588 5
8503 8589 5
8504 8590 5
8505 8591 5
8506 8592 5
8507 8593 5
8508 8594 5
8509 8595 5
8510 8596 5
8511 8597 5
8512 8598 5
8513 8599 5
8514 8600 5
8515 8601 5
8516 8602 5
8517 8603 5
8518 8604 5
8519 8605 5
8520 8606 5
8521 8607 5
8522 8608 5
8523 8609 5
8524 8610 5
8525 8611 5
8526 8612 5
8527 8613 5
8528 8614 5
8529 8615 5
8530 8616 4
8531 8617 4
8532 8618 3
8533 8619 3
8534 8620 3
8535 8621 3
8536 8622 3
8537 8623 3
8538 8624 4
8539 8625 4
8540 8626 4
8541 8627 4
8542 8628 3
8543 8629 3
8544 8630 3
8545 8631 3
8546 8632 3
8547 8633 3
8548 8634 4
8549 8635 4
8550 8636 4
8551 8637 4
8552 8638 4
8553 8639 4
8554 8640 3
```



```
TARG_POS = %BPVAL - .BITS_TO_ROT;
SRC_POS = 0;
SIZ = .BITS_TO_ROT;
(SECOND_CELL)<.TARG_POS, .SIZ> = .OPERAND<.SRC_POS, .SIZ>;
```

```
! Move the high order bits of the source to the low order
! bits of the target.
```

```
TARG_POS = 0;
SRC_POS = .BITS_TO_ROT;
SIZ = %BPVAL - .BITS_TO_ROT;
(SECOND_CELL)<.TARG_POS, .SIZ> = .OPERAND<.SRC_POS, .SIZ>;
```

```
! Adjust the stack pointer,
```

```
POP(1);
INSTRUC = .INSTRUC + 1;
END;
```

```
END;
```

```
! COPY The top stack cell is PUSHed
```

```
[DST$K STK_OP]:
BEGIN
PUSH(1);
TOP_CELL = .SECOND_CELL;
INSTRUC = .INSTRUC + 1;
END;
```

```
! EXCHANGE The top two stack cells are exchanged
```

```
[DST$K STK_EXCH]:
BEGIN
LOCAL WORK_CELL;
WORK_CELL = .TOP_CELL;
TOP_CELL = .SECOND_CELL;
SECOND_CELL = .WORK_CELL;
INSTRUC = .INSTRUC + 1;
END;
```

8555 8641
8556 8642
8557 8643
8558 8644
8559 8645
8560 8646
8561 8647
8562 8648
8563 8649
8564 8650
8565 8651
8566 8652
8567 8653
8568 8654
8569 8655
8570 8656
8571 8657
8572 8658
8573 8659
8574 8660
8575 8661
8576 8662
8577 8663
8578 8664
8579 8665
8580 8666
8581 8667
8582 8668
8583 8669
8584 8670
8585 8671
8586 8672
8587 8673
8588 8674
8589 8675
8590 8676
8591 8677
8592 8678
8593 8679
8594 8680
8595 8681
8596 8682
8597 8683
8598 8684
8599 8685
8600 8686
8601 8687
8602 8688
8603 8689
8604 8690
8605 8691
8606 8692
8607 8693
8608 8694
8609 8695
8610 8696
8611 8697

STORE BYTE WORD OR LONG

The byte following this operand is interpreted as a signed (for consistency with something, see Grove) byte offset into the stack. The low order byte, word, or longword of the top stack cell is copied into the byte, word or longword at this location:

address of the second stack cell
+
the specified byte offset.

(Keep in mind that the address of the third stack cell is the address of the second stack cell plus four.)
The stack is popped.

```
[DST$K_STK_STO_B,  
DST$K_STK_STO_W,  
DST$K_STK_STO_L]:  
  BEGIN  
    LOCAL TARGET, SIZ;  
    TARGET = SECOND_CELL + .INSTRUC[1];  
    SIZ =  
      (CASE .INSTRUC[0] FROM DST$K_STK_STO_B TO DST$K_STK_STO_L OF  
        SET  
          [DST$K_STK_STO_B]: 1;  
          [DST$K_STK_STO_W]: 2;  
          [DST$K_STK_STO_L]: 4;  
        TES);  
    CHSMOVE(.SIZ, TOP_CELL, .TARGET);  
    POP(1);  
    INSTRUC = .INSTRUC + 2;  
  END;
```

POP The top stack cell is removed from the stack.

```
[DST$K_STK_POP]:  
  BEGIN  
    POP(1);  
    INSTRUC = .INSTRUC + 1;  
  END;
```

RTNCALL

Call a compiler-supplied routine to compute a value to be put on the stack. We assume that the routine address is already on top of the stack. That address is popped and the returned value is pushed on the stack.

```
[DST$K_STK_RTNCALL]:
```

```

: 8612      8698 4
: 8613      8699 4
: 8614      8700 4
: 8615      8701 4
: 8616      8702 4
: 8617      8703 4
: 8618      8704 4
: 8619      8705 4
: 8620      8706 4
: 8621      8707 4
: 8622      8708 4
: 8623      8709 4
: 8624      8710 3
: 8625      8711 3
: 8626      8712 3
: 8627      8713 3
: 8628      8714 3
: 8629      8715 3
: 8630      8716 3
: 8631      8717 3
: 8632      8718 3
: 8633      8719 4
: 8634      8720 4
: 8635      8721 4
: 8636      8722 4
: 8637      8723 4
: 8638      8724 4
: 8639      8725 4
: 8640      8726 4
: 8641      8727 4
: 8642      8728 4
: 8643      8729 4
: 8644      8730 4
: 8645      8731 3
: 8646      8732 3
: 8647      8733 3
: 8648      8734 3
: 8649      8735 3
: 8650      8736 3
: 8651      8737 3
: 8652      8738 3
: 8653      8739 3
: 8654      8740 3
: 8655      8741 3
: 8656      8742 4
: 8657      8743 4
: 8658      8744 4
: 8659      8745 4
: 8660      8746 4
: 8661      8747 4
: 8662      8748 4
: 8663      8749 4
: 8664      8750 4
: 8665      8751 4
: 8666      8752 4
: 8667      8753 4
: 8668      8754 4

      BEGIN
      LOCAL
      Temp_thunk_addr;
      Temp_thunk_addr = .TOP_CELL;
      CALL_RESULT = 0;
      POP(T);
      VALSPEC_ROUT_CALL(CALL_RESULT, .Temp_thunk_addr, FALSE, TRUE,
      .STACK_PTR, .UNDERFLOW_POINT-.STACK_PTR);
      PUSH(1);
      TOP_CELL = .CALL_RESULT;
      FRAMEPTR[0] = .DBG$REG_VALUES[13];
      INSTRUC = .INSTRUC + 1;
      END;

: RTN_NOFP
:
: Call a compiler-supplied routine to
: compute a value to be put on the stack.
: is pushed on the stack. Same as RTNCALL
: except no FP is passed in to thunk.

[DESTK_STK_RTN_NOFP]:
      BEGIN
      LOCAL
      Temp_thunk_addr;
      Temp_thunk_addr = .TOP_CELL;
      CALL_RESULT = 0;
      POP(T);
      VALSPEC_ROUT_CALL(CALL_RESULT, .Temp_thunk_addr, FALSE, FALSE,
      .STACK_PTR, .UNDERFLOW_POINT-.STACK_PTR);
      PUSH(1);
      TOP_CELL = .CALL_RESULT;
      FRAMEPTR[0] = .DBG$REG_VALUES[13];
      INSTRUC = .INSTRUC + 1;
      END;

: RTNCALL_ALT
:
: Call a compiler-supplied routine to
: compute a value to be put on the stack.
: We assume that the routine address is
: already on top of the stack. That ad-
: dress is popped and the returned value
: (a quadword) is pushed on the stack.

[DESTK_STK_RTNCALL_ALT]:
      BEGIN
      LOCAL
      Temp_thunk_addr;
      CALL_RESULT = VECTOR[ 4 ];
      Temp_thunk_addr = .TOP_CELL;
      CALL_RESULT[0] = 0;
      CALL_RESULT[1] = 0;
      CALL_RESULT[2] = 0;
      CALL_RESULT[3] = 0;
      POP(T);
      VALSPEC_ROUT_CALL(CALL_RESULT, .Temp_thunk_addr, TRUE, TRUE,
      .STACK_PTR, .UNDERFLOW_POINT-.STACK_PTR);
      PUSH( 4 );

      ! Save the routine address,
      ! Pop off the thunk address
      ! Push for the call result
      ! Pop off the thunk address
```

```

: 8669      8755      4      CH$MOVE( 16, CH$PTR( CALL RESULT ), CH$PTR( .STACK_PTR ) );
: 8670      8756      4      FRAMEPTR[0] = .DBG$REG_VALUES[13];
: 8671      8757      4      INSTRUC = .INSTRUC + 1;
: 8672      8758      3      END;
: 8673      8759      3
: 8674      8760      3
: 8675      8761      3      : PUSH_OUTER_REC      Push the start address of the outer most
: 8676      8762      3      :                      record, described by the primary pointed
: 8677      8763      3      :                      to by DBG$GL CURRENT PRIMARY. Originally
: 8678      8764      3      :                      implemented for use by languages which allow
: 8679      8765      3      :                      Self-referential records, (PL/I, ADA)
: 8680      8766      3
: 8681      8767      3      :                      Self-referential records are those which
: 8682      8768      3      :                      contain fields or structures whose actual
: 8683      8769      3      :                      allocated length depends on some preceding
: 8684      8770      3      :                      value within the record. Thus, the address
: 8685      8771      3      :                      of any fields following the field or structure
: 8686      8772      3      :                      is not known at compile time, and therefore must
: 8687      8773      3      :                      be calculated at run-time.
: 8688      8774      3
: 8689      8775      3      [DST$K_STK_PUSH_OUTER_REC]:
: 8690      8776      4      BEGIN
: 8691      8777      4      PUSH(1);
: 8692      8778      4      TOP_CELL = DBG$GET_OUTER_REC_ADDRESS(.DBG$GL_CURRENT_PRIMARY);
: 8693      8779      4      INSTRUC = .INSTRUC + 1;
: 8694      8780      3      END;
: 8695      8781      3
: 8696      8782      3
: 8697      8783      3      : PUSH_INNER_REC      Push the start address of the inner most
: 8698      8784      3      :                      record, described by the primary pointed
: 8699      8785      3      :                      to by DBG$GL CURRENT PRIMARY. Originally
: 8700      8786      3      :                      implemented for use by languages which allow
: 8701      8787      3      :                      Self-referential records, (PL/I, ADA)
: 8702      8788      3
: 8703      8789      3      [DST$K_STK_PUSH_INNER_REC]:
: 8704      8790      4      BEGIN
: 8705      8791      4      PUSH(1);
: 8706      8792      4      TOP_CELL = DBG$GET_INNER_REC_ADDRESS(.DBG$GL_CURRENT_PRIMARY);
: 8707      8793      4      INSTRUC = .INSTRUC + 1;
: 8708      8794      3      END;
: 8709      8795      3
: 8710      8796      3
: 8711      8797      3      : Any other op-code is an error. Signal an internal bug.
: 8712      8798      3
: 8713      8799      3      [INRANGE, OUTRANGE]:
: 8714      8800      3      $DBG_ERROR('RSTACCESS\STACK_MACHINE - Invalid stack machine opcode. Bad DST');
: 8715      8801      3
: 8716      8802      3      TES;
: 8717      8803      3
: 8718      8804      2      END;      ! End of WHILE loop over instructions
: 8719      8805      2
: 8720      8806      2
: 8721      8807      2      ! Fill in the result address and return.
: 8722      8808      2      RESULT_PTR[0] = TOP_CELL;
: 8723      8809      2      RETURN;
: 8724      8810      2
: 8725      8811      2
```

```
.PSECT DBG$PLIT,NOWRT, SHR, PIC,0

43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00561 P.AEJ: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 10\
30 31 20 45 4E 49 48 43 41 4D 5F 4B 00570
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 0057C P.AEK: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 10\
30 31 20 45 4E 49 48 43 41 4D 5F 4B 00588
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00597 P.AEL: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 10\
30 31 20 45 4E 49 48 43 41 4D 5F 4B 005A6
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 005B2 P.AEM: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 10\
30 31 20 45 4E 49 48 43 41 4D 5F 4B 005C1
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 005CD P.AEN: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 30\
30 33 20 45 4E 49 48 43 41 4D 5F 4B 005DC
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 005EB P.AEO: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 10\
30 31 20 45 4E 49 48 43 41 4D 5F 4B 005F7
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00603 P.AEP: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 10\
30 31 20 45 4E 49 48 43 41 4D 5F 4B 00612
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 0061E P.AEQ: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 40\
30 34 20 45 4E 49 48 43 41 4D 5F 4B 0062D
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00639 P.AER: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 00648
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00654 P.AES: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 40\
30 34 20 45 4E 49 48 43 41 4D 5F 4B 00663
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 0066F P.AFT: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 0067E
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 0068A P.AEU: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 40\
30 34 20 45 4E 49 48 43 41 4D 5F 4B 00699
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 006A5 P.AEV: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 006B4
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 006C0 P.AEW: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 40\
30 34 20 45 4E 49 48 43 41 4D 5F 4B 006CF
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 006DB P.AEX: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 50\
30 35 20 45 4E 49 48 43 41 4D 5F 4B 006EA
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 006F6 P.AEY: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 00705
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00711 P.AEZ: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 40\
30 34 20 45 4E 49 48 43 41 4D 5F 4B 00720
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 0072C P.AFA: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 0073B
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00747 P.AFB: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 00756
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00762 P.AFC: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 00771
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 0077D P.AFD: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 40\
30 34 20 45 4E 49 48 43 41 4D 5F 4B 0078C
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00798 P.AFE: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 007A7
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 007B3 P.AFF: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 007C2
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 007CE P.AFG: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 10\
30 31 20 45 4E 49 48 43 41 4D 5F 4B 007DD
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 007E9 P.AFH: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
30 32 20 45 4E 49 48 43 41 4D 5F 4B 007F8
43 41 54 53 5C 53 53 45 43 43 41 54 53 52 1A 00804 P.AFI: .ASCII <26>\RSTACCESS\<92>\STACK_MACHINE 20\
```

Page 264
(48)

```

8 6
16-Sep-1984 02:48:17 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:18:26 [DEBUG.SRC]RSTACKACCESS.B32;1

```

Page 265
(48)

RS
VO[illegible]

50	01	DO	000D7	9\$:	MOVL	#1	REGISTR	:		
	44	11	000DA		BRB	24\$:		
50	02	DO	000DC	10\$:	MOVL	#2	REGISTR	:		
	3F	11	000DF		BRB	24\$:		
50	03	DO	000E1	11\$:	MOVL	#3	REGISTR	:		
	3A	11	000E4		BRB	24\$:		
50	04	DO	000E6	12\$:	MOVL	#4	REGISTR	:		
	35	11	000E9		BRB	24\$:		
50	05	DO	000EB	13\$:	MOVL	#5	REGISTR	:		
	30	11	000EE		BRB	24\$:		
50	06	DO	000FC	14\$:	MOVL	#6	REGISTR	:		
	2B	11	000F3		BRB	24\$:		
50	07	DO	000F5	15\$:	MOVL	#7	REGISTR	:		
	26	11	000F8		BRB	24\$:		
50	08	DO	000FA	16\$:	MOVL	#8	REGISTR	:		
	21	11	000FD		BRB	24\$:		
50	09	DO	000FF	17\$:	MOVL	#9	REGISTR	:		
	1C	11	00102		BRB	24\$:		
50	0A	DO	00104	18\$:	MOVL	#10	REGISTR	:		
	17	11	00107		BRB	24\$:		
50	0B	DO	00109	19\$:	MOVL	#11	REGISTR	:		
	12	11	0010C		BRB	24\$:		
50	0C	DO	0010E	20\$:	MOVL	#12	REGISTR	:		
	0D	11	00111		BRB	24\$:		
50	0D	DO	00113	21\$:	MOVL	#13	REGISTR	:		
	08	11	00116		BRB	24\$:		
50	0E	DO	00118	22\$:	MOVL	#14	REGISTR	:		
	03	11	0011B		BRB	24\$:		
50	0F	DO	0011D	23\$:	MOVL	#15	REGISTR	:		
	00000000G	00	00120	24\$:	TSTL	DBG\$REG_VECTOR[REGISTR]		:	8276	
		0C	13	00127	BEQL	25\$:		
66	00000000G	00	00129		MOVL	DBG\$REG_VALUES[REGISTR], (STACK_PTR)		:	8279	
		57	D6	00131	INCL	INSTRUC		:	8280	
		05	11	00133	BRB	26\$:	8276	
0000V	CF	00	FB	00135	25\$:	CALLS	#0, VALSPEC_SCOPE_ERROR	:	8284	
OC	BC	00	DO	0013A	26\$:	MOVL	DBG\$REG_VALUES+52, @FRAMEPTR	:	8286	
		75	11	00142	BRB	33\$:	8228	
56		04	C2	00144	27\$:	SUBL2	#4, STACK_PTR	:	8297	
5A		56	D1	00147	CMPL	STACK_PTR, OVERFLOW_POINT		:		
		15	1E	0014A	BGEQU	28\$:		
	00000000'	EF	9F	0014C	PUSHAB	P.AEK		:		
		01	DD	00152	PUSHL	#1		:		
	00028362	8F	DD	00154	PUSHL	#164706		:		
00000000G	00	03	FB	0015A	CALLS	#3, LIB\$SIGNAL		:		
	50	01	A7	9E	00161	28\$:	MOVAB	1(R7), OPERAND	:	8298
	66	60	98	00165	CVTBL	(OPERAND), (STACK_PTR)		:	8299	
		037E	31	00168	BRW	87\$:	8300	
	56	04	C2	0016B	29\$:	SUBL2	#4, STACK_PTR	:	8306	
	5A	56	D1	0016E	CMPL	STACK_PTR, OVERFLOW_POINT		:		
		15	1E	00171	BGEQU	30\$:		
	00000000'	EF	9F	00173	PUSHAB	P.AEL		:		
		01	DD	00179	PUSHL	#1		:		
	00028362	8F	DD	0017B	PUSHL	#164706		:		
00000000G	00	03	FB	00181	CALLS	#3, LIB\$SIGNAL		:		
	50	01	A7	9E	00188	30\$:	MOVAB	1(R7), OPERAND	:	8307
	66	60	32	0018C	CVTBL	(OPERAND), (STACK_PTR)		:	8308	
		00B7	31	0018F	BRW	43\$:	8309	

		56	04	C2	00192	31\$:	SUBL2	#4, STACK_PTR	:	8315	
		5A	56	D1	00195		CMPL	STACK_PTR, OVERFLOW_POINT	:		
			15	1E	00198		BGEQU	32\$:		
			EF	9F	0019A		PUSHAB	P.AEM	:		
			01	DD	001A0		PUSHL	#1	:		
			8F	DD	001A2		PUSHL	#164706	:		
		00000000G	00	03	FB	001A8	CALLS	#3, LIB\$SIGNAL	:		
			50	01	A7	9E	001AF	32\$:	MOVAB	1(R7), OPERAND	8316
			66		60	D0	001B3		MOVL	(OPERAND), (STACK_PTR)	8317
			57		05	C0	001B6		ADDL2	#5, INSTRUC	8318
				44	11	001B9	33\$:	BRB	38\$:	8228
			58	01	A7	9A	001BB	34\$:	MOVZBL	1(INSTRUC), R8	8329
			58		01	7A	001BF		EMUL	#1, R8, #0, -(SP)	
7E	00		8E		04	7B	001C4		EDIV	#4, (SP)+, R0, R0	
50	50				50	D5	001C9		TSTL	R0	
					06	13	001CB		BEQL	35\$	
		50	04		50	C3	001CD		SUBL3	R0, #4, R0	
					02	11	001D1		BRB	36\$	
					50	D4	001D3	35\$:	CLRL	R0	
			50		58	C0	001D5	36\$:	ADDL2	R8, R0	
			56		50	C2	001D8		SUBL2	R0, STACK_PTR	
			5A		56	D1	001DB		CMPL	STACK_PTR, OVERFLOW_POINT	
					15	1E	001DE		BGEQU	37\$	
					EF	9F	001E0		PUSHAB	P.AEN	
					01	DD	001E6		PUSHL	#1	
					8F	DD	001E8		PUSHL	#164706	
		00000000G	00	03	FB	001EE	CALLS	#3, LIB\$SIGNAL	:		
66	02		A7	58	28	001F5	37\$:	MOV C3	R8, 2(INSTRUC), (STACK_PTR)	8330	
			57	02	A8	47	9E	001FA	MOVAB	2(R8)[INSTRUC], INSTRU	8331
					4B	11	001FF	38\$:	BRB	44\$	8228
			56		04	C2	00201	39\$:	SUBL2	#4, STACK_PTR	8341
			5A		56	D1	00204		CMPL	STACK_PTR, OVERFLOW_POINT	
					15	1E	00207		BGEQU	40\$	
					EF	9F	00209		PUSHAB	P.AEO	
					01	DD	0020F		PUSHL	#1	
					8F	DD	00211		PUSHL	#164706	
		00000000G	00	03	FB	00217	CALLS	#3, LIB\$SIGNAL	:		
			66	01	A7	9A	0021E	40\$:	MOVZBL	1(INSTRUC), (STACK_PTR)	8342
				02	C4	31	00222		BRW	87\$	8343
			56		04	C2	00225	41\$:	SUBL2	#4, STACK_PTR	8349
			5A		56	D1	00228		CMPL	STACK_PTR, OVERFLOW_POINT	
					15	1E	0022B		BGEQU	42\$	
					EF	9F	0022D		PUSHAB	P.AEP	
					01	DD	00233		PUSHL	#1	
					8F	DD	00235		PUSHL	#164706	
		00000000G	00	03	FB	0023B	CALLS	#3, LIB\$SIGNAL	:		
			50	01	A7	9E	00242	42\$:	MOVAB	1(R7), OPERAND	8350
			66		60	3C	00246		MOVZWL	(OPERAND), (STACK_PTR)	8351
			57		03	C0	00249	43\$:	ADDL2	#3, INSTRUC	8352
				F	D4	31	0024C	44\$:	BRW	1\$	8228
			50		66	D0	0024F	45\$:	MOVL	(STACK_PTR), OPERAND	8364
			66		60	98	00252		CVTBL	(OPERAND), (STACK_PTR)	8365
					1E	11	00255		BRB	50\$	8366
			50		66	D0	00257	46\$:	MOVL	(STACK_PTR), OPERAND	8372
			66		60	32	0025A		CVTWL	(OPERAND), (STACK_PTR)	8373
					16	11	0025D		BRB	50\$	8374
			50		66	D0	0025F	47\$:	MOVL	(STACK_PTR), OPERAND	8380

66	60	DO	00262	MOVL	(OPERAND), (STACK_PTR)	8381	
	0E	11	00265	BRB	50\$	8382	
50	66	DO	00267	48\$: MOVL	(STACK_PTR), OPERAND	8394	
66	60	9A	0026A	MOVZBL	(OPERAND), (STACK_PTR)	8395	
	06	11	0026D	BRB	50\$	8396	
50	66	DO	0026F	49\$: MOVL	(STACK_PTR), OPERAND	8402	
66	60	3C	00272	MOVZWL	(OPERAND), (STACK_PTR)	8403	
	03E8	31	00275	50\$: BRW	106\$	8404	
50	08	A6	9E	00278	51\$: MOVAB	8(R6), RO	8414
50		59	D1	0027C	CMPL	UNDERFLOW_POINT, RO	
		15	18	0027F	BGEQ	52\$	
	00000000'	EF	9F	00281	PUSHAB	P.AEQ	
		01	DD	00287	PUSHL	#1	
00000000G		8F	DD	00289	PUSHL	#164706	
00		03	FB	0028F	CALLS	#3, LIB\$SIGNAL	
66		86	C0	00296	52\$: ADDL2	(STACK_PTR)+, (STACK_PTR)	8415
59		56	D1	00299	CMPL	STACK_PTR, UNDERFLOW_POINT	8416
		D7	1B	0029C	BLEQU	50\$	
	00000000'	EF	9F	0029E	PUSHAB	P.AER	
		5C	11	002A4	BRB	57\$	
50	08	A6	9E	002A6	53\$: MOVAB	8(R6), RO	8427
50		59	D1	002AA	CMPL	UNDERFLOW_POINT, RO	
		15	18	002AD	BGEQ	54\$	
	00000000'	EF	9F	002AF	PUSHAB	P.AES	
		01	DD	002B5	PUSHL	#1	
00000000G		8F	DD	002B7	PUSHL	#164706	
66	00	03	FB	002BD	CALLS	#3, LIB\$SIGNAL	
86	04	A6	C3	002C4	54\$: SUBL3	4(STACK_PTR), (STACK_PTR)+, (STACK_PTR)	8428
59		56	D1	002C9	CMPL	STACK_PTR, UNDERFLOW_POINT	8429
		A7	1B	002CC	BLEQU	50\$	
	00000000'	EF	9F	002CE	PUSHAB	P.AET	
		79	11	002D4	BRB	63\$	
50	08	A6	9E	002D6	55\$: MOVAB	8(R6), RO	8440
50		59	D1	002DA	CMPL	UNDERFLOW_POINT, RO	
		15	18	002DD	BGEQ	56\$	
	00000000'	EF	9F	002DF	PUSHAB	P.AEU	
		01	DD	002E5	PUSHL	#1	
00000000G		8F	DD	002E7	PUSHL	#164706	
00		03	FB	002ED	CALLS	#3, LIB\$SIGNAL	
66		86	C4	002F4	56\$: MULL2	(STACK_PTR)+, (STACK_PTR)	8441
59		56	D1	002F7	CMPL	STACK_PTR, UNDERFLOW_POINT	8442
		4B	1B	002FA	BLEQU	62\$	
	00000000'	EF	9F	002FC	PUSHAB	P.AEV	
		4B	11	00302	DRB	63\$	
50	08	A6	9E	00304	57\$: MOVAB	8(R6), RO	8453
50		59	D1	00308	58\$: CMPL	UNDERFLOW_POINT, RO	
		15	18	0030B	BGEQ	59\$	
	00000000'	EF	9F	0030D	PUSHAB	P.AEW	
		01	DD	00313	PUSHL	#1	
00000000G		8F	DD	00315	PUSHL	#164706	
00		03	FB	0031B	CALLS	#3, LIB\$SIGNAL	
	04	A6	D5	00322	59\$: TSTL	4(STACK_PTR)	8454
		18	12	00325	BNEQ	61\$	
	00000000'	EF	9F	00327	PUSHAB	P.AEX	8456
		01	DD	0032D	60\$: PUSHL	#1	
00000000G		8F	DD	0032F	PUSHL	#164706	
00		03	FB	00335	CALLS	#3, LIB\$SIGNAL	

66	86	04	FCE4	31	0033C	BRW	1\$		
	59		A6	C7	0033F	DIVL3	4(STACK_PTR), (STACK_PTR)+, (STACK_PTR)	8460	
			56	D1	00344	CMPL	STACK_PTR, UNDERFLOW_POINT	8461	
			52	1B	00347	BLEQU	68\$		
		00000000'	EF	9F	00349	PUSHAB	P.AEY		
			7E	11	0034F	BRB	71\$		
	50	08	A6	9E	0C351	MOVAB	8(R6), R0	8474	
	50		59	D1	00355	CMPL	UNDERFLOW_POINT, R0		
			15	1B	00358	BGEQ	65\$		
		00000000'	EF	9F	0035A	PUSHAB	P.AEZ		
			01	DD	00360	PUSHL	#1		
		00028362	8F	DD	00362	PUSHL	#164706		
00000000G	00		03	FB	00368	CALLS	#3, LIB\$SIGNAL	8475	
	50		66	D0	0036F	MOVL	(STACK_PTR), R0		
			03	1B	00372	BGEQ	66\$		
	50		50	CE	00374	MNEGL	R0, R0		
	20		50	D1	00377	CMPL	R0, #32	66\$:	
			22	19	0037A	BLSS	69\$		
	56		04	C0	0037C	ADDL2	#4, STACK_PTR	8478	
	59		56	D1	0037F	CMPL	STACK_PTR, UNDERFLOW_POINT		
			15	1B	00382	BLEQU	67\$		
		00000000'	EF	9F	00384	PUSHAB	P.AFA		
			01	DD	0038A	PUSHL	#1		
		00028362	8F	DD	0038C	PUSHL	#164706		
00000000G	00		03	FB	00392	CALLS	#3, LIB\$SIGNAL	8479	
			66	D4	00399	CLRL	(STACK_PTR)	8475	
			02C2	31	0039B	BRW	106\$	8484	
			66	D5	0039E	TSTL	(STACK_PTR)		
			16	15	003A0	BLEQ	70\$		
03	A6	03	A6	78	003A2	ASHL	(STACK_PTR)+, 3(STACK_PTR), 3(STACK_PTR)	8490	
	56		03	C0	003A8	ADDL2	#3, STACK_PTR	8491	
	59		56	D1	003AB	CMPL	STACK_PTR, UNDERFLOW_POINT		
			EB	1B	003AE	BLEQU	68\$		
		00000000'	EF	9F	003B0	PUSHAB	P.AFB		
			17	11	003B6	BRB	71\$		
	51		86	CE	003B8	MNEGL	(STACK_PTR)+, POSITION	8503	
	20		51	C3	003BB	SUBL3	POSITION, #32, SIZ	8504	
66	66		51	EF	003BF	EXTZV	POSITION, SIZ, (STACK_PTR), (STACK_PTR)	8505	
	59		56	D1	003C4	CMPL	STACK_PTR, UNDERFLOW_POINT	8506	
			D2	1B	003C7	BLEQU	68\$		
		00000000'	EF	9F	003C9	PUSHAB	P.AFC		
			6D	11	003CF	BRB	74\$		
	50	08	A6	9E	003D1	MOVAB	8(R6), R0	8521	
	50		59	D1	003D5	CMPL	UNDERFLOW_POINT, R0		
			15	1B	003D8	BGEQ	73\$		
		00000000'	EF	9F	003DA	PUSHAB	P.AFD		
			01	DD	003E0	PUSHL	#1		
		00028362	8F	DD	003E2	PUSHL	#164706		
00000000G	00		03	FB	003E8	CALLS	#3, LIB\$SIGNAL	8522	
	66		01	7A	003EF	EMUL	#1, (STACK_PTR), #0, -(SP)		
7E	50		20	7B	003F4	EDIV	#32, (SP)+, BITS_TO_ROT, BITS_TO_ROT	8530	
	54	04	A6	9E	003F9	MOVAB	4(STACK_PTR), R4	8560	
	53		50	C3	003FD	SUBL3	BITS_TO_ROT, #32, R3	8523	
			50	D5	00401	TSTL	BITS_TO_ROT		
			3B	15	00403	BLEQ	75\$		
	58		64	D0	00405	MOVL	(R4), OPERAND	8530	
	6E		50	D0	00408	MOVL	BITS_TO_ROT, TARG_POS	8551	

			55	D4	0040B	CLRL	SRC_POS	8552	
		51	AE	9E	0040D	MOVAB	TARG_POS-32, SIZ	8553	
		51	51	CE	00411	MNEGL	SIZ, SIZ		
52	58	51	55	EF	00414	EXTZV	SRC_POS, SIZ, OPEPAND, R2	8554	
64	51	6E	52	F0	00419	INSV	R2, TARG_POS, SIZ, (R4)		
			6E	D4	0041E	CLRL	TARG_POS	8559	
		55	53	D0	00420	MOVL	R3, SRC_POS	8560	
		51	50	D0	00423	MOVL	BITS_TO_ROT, SIZ	8561	
50	58	51	55	EF	00426	EXTZV	SRC_POS, SIZ, OPERAND, R0	8562	
64	51	6E	50	F0	0042B	INSV	R0, TARG_POS, SIZ, (R4)		
		56	04	C0	00430	ADDL2	#4, STACK_PTR	8566	
		59	56	D1	00433	CMPL	STACK_PTR, UNDERFLOW_POINT		
			6E	1B	00436	BLEQU	80\$		
			EF	9F	00438	PUSHAB	P.AFE		
			35	11	0043E	BRB	76\$		
		58	64	D0	00440	MOVL	(R4), OPERAND	8576	
		58	53	D0	00443	MOVL	R3, TARG_POS	8597	
			55	D4	00446	CLRL	SRC_POS	8598	
		51	50	D0	00448	MOVL	BITS_TO_ROT, SIZ	8599	
52	58	51	55	EF	0044B	EXTZV	SRC_POS, SIZ, OPERAND, R2	8600	
64	51	58	52	F0	00450	INSV	R2, TARG_POS, SIZ, (R4)		
			58	D4	00455	CLRL	TARG_POS	8606	
		55	50	D0	00457	MOVL	BITS_TO_ROT, SRC_POS	8607	
		51	53	D0	0045A	MOVL	R3, SIZ	8608	
52	58	51	55	EF	0045D	EXTZV	SRC_POS, SIZ, OPERAND, R2	8609	
64	51	58	52	F0	00462	INSV	R2, TARG_POS, SIZ, (R4)		
		56	04	C0	00467	ADDL2	#4, STACK_PTR	8614	
		59	56	D1	0046A	CMPL	STACK_PTR, UNDERFLOW_POINT		
			37	1B	0046D	BLEQU	80\$		
			EF	9F	0046F	PUSHAB	P.AFF		
			0085	31	00475	BRW	89\$		
		56	04	C2	00478	SUBL2	#4, STACK_PTR	8625	
		5A	56	D1	0047B	CMPL	STACK_PTR, OVERFLOW_POINT		
			15	1E	0047E	BGEQU	78\$		
			EF	9F	00480	PUSHAB	P.AFG		
			01	DD	00486	PUSHL	#1		
			8F	DD	00488	PUSHL	#164706		
		00000000G	00	03	FB	CALLS	#3, LIB\$SIGNAL		
			66	A6	D0	00495	MOVL	4(STACK_PTR), (STACK_PTR)	8626
			71	11	00499	BRB	90\$	8627	
		50	66	D0	0049B	MOVL	(STACK_PTR), WORK_CELL	8636	
		66	04	A6	D0	0049E	MOVL	4(STACK_PTR), (STACK_PTR)	8637
	04	A6	50	D0	004A2	MOVL	WORK_CELL, 4(STACK_PTR)	8638	
			64	11	004A6	BRB	90\$	8639	
		50	01	A7	9A	004A8	MOVZBL	1(INSTRUC), R0	8666
		51	04	A046	9E	004AC	MOVAB	4(R0)[STACK_PTR], TARGET	
	02	24	67	8F	004B1	CASEB	(INSTRUC), #36, #2	8668	
0010		000B	0006		004B5	.WORD	83\$-82\$,-		
							84\$-82\$,-		
							85\$-82\$,-		
		50	01	D0	004BB	MOVL	#1, SIZ		
			08	11	004BE	BRB	86\$		
		50	02	D0	004C0	MOVL	#2, SIZ		
			03	11	004C3	BRB	86\$		
		50	04	D0	004C5	MOVL	#4, SIZ		
	61	66	50	28	004C8	MOVCS	SIZ, (STACK_PTR), (TARGET)	8674	
		56	04	C0	004CC	ADDL2	#4, STACK_PTR	8675	

	59		56	D1	004CF	CMPL	STACK_PTR, UNDERFLOW_POINT		
		00000000'	15	1B	004D2	BLEQU	87\$		
			EF	9F	004D4	PUSHAB	P.AFH		
		00028362	01	DD	004DA	PUSHL	#1		
00000000G	00		8F	DD	004DC	PUSHL	#164706		
	57		03	FB	004E2	CALLS	#3, LIB\$SIGNAL		
			02	CO	004E9	ADDL2	#2, INSTRU	8676	
			FB34	31	004EC	BRW	1\$	8228	
	56		04	CO	004EF	ADDL2	#4, STACK_PTR	8685	
	59		56	D1	004F2	CMPL	STACK_PTR, UNDERFLOW_POINT		
		00000000'	15	1B	004F5	BLEQU	90\$		
			EF	9F	004F7	PUSHAB	P.AFI		
		00028362	01	DD	004FD	PUSHL	#1		
00000000G	00		8F	DD	004FF	PUSHL	#164706		
			03	FB	00505	CALLS	#3, LIB\$SIGNAL		
	52		0151	31	0050C	BRW	106\$	8696	
		04	86	DO	0050F	MOVL	(STACK_PTR)+, TEMP_THUNK_ADDR	8701	
	59		AE	D4	00512	CLRL	CALL_RESULT	8702	
			56	D1	00515	CMPL	STACK_PTR, UNDERFLOW_POINT	8703	
		00000000'	15	1B	00518	BLEQU	92\$		
			EF	9F	0051A	PUSHAB	P.AFJ		
		00028362	01	DD	00520	PUSHL	#1		
00000000G	00		8F	DD	00522	PUSHL	#164706		
7E	59		03	FB	00528	CALLS	#3, LIB\$SIGNAL		
			56	C3	0052F	SUBL3	STACK_PTR, UNDERFLOW_POINT, -(SP)	8705	
			56	DD	00533	PUSHL	STACK_PTR		
			01	DD	00535	PUSHL	#1	8704	
			7E	D4	00537	CLRL	-(SP)		
		18	52	DD	00539	PUSHL	TEMP_THUNK_ADDR		
			AE	9F	0053B	PUSHAB	CALL_RESULT		
0000V	CF		06	FB	0053E	CALLS	#6, VALSPEC_ROUT_CALL		
	56		04	C2	00543	SUBL2	#4, STACK_PTR	8706	
	5A		56	D1	00546	CMPL	STACK_PTR, OVERFLOW_POINT		
		00000000'	57	1E	00549	BGEQU	96\$		
			EF	9F	0054B	PUSHAB	P.AFK		
			40	11	00551	BRB	95\$		
	52		86	DO	00553	MOVL	(STACK_PTR)+, TEMP_THUNK_ADDR	8722	
		04	AE	D4	00556	CLRL	CALL_RESULT	8723	
	59		56	D1	00559	CMPL	STACK_PTR, UNDERFLOW_POINT	8724	
		00000000'	15	1B	0055C	BLEQU	94\$		
			EF	9F	0055E	PUSHAB	P.AFL		
		00028362	01	DD	00564	PUSHL	#1		
00000000G	00		8F	DD	00566	PUSHL	#164706		
7E	59		03	FB	0056C	CALLS	#3, LIB\$SIGNAL		
			56	C3	00573	SUBL3	STACK_PTR, UNDERFLOW_POINT, -(SP)	8726	
			56	DD	00577	PUSHL	STACK_PTR		
			7E	7C	00579	CLRL	-(SP)	8725	
		18	52	DD	0057B	PUSHL	TEMP_THUNK_ADDR		
			AE	9F	0057D	PUSHAB	CALL_RESULT		
0000V	CF		06	FB	00580	CALLS	#6, VALSPEC_ROUT_CALL		
	56		04	C2	00585	SUBL2	#4, STACK_PTR	8727	
	5A		56	D1	00588	CMPL	STACK_PTR, OVERFLOW_POINT		
		00000000'	15	1E	0058B	BGEQU	96\$		
			EF	9F	0058D	PUSHAB	P.AFM		
		00028362	01	DD	00593	PUSHL	#1		
00000000G	00		8F	DD	00595	PUSHL	#164706		
			03	FB	0059B	CALLS	#3, LIB\$SIGNAL		

66	04	AE	D0	005A2	96\$:	MOVL	CALL_RESULT, (STACK_PTR)	8728	
		59	11	005A6		BRB	100\$	8729	
52		86	D0	005A8	97\$:	MOVL	(STACK_PTR)+, TEMP_THUNK_ADDR	8746	
	08	AE	7C	005AB		CLRQ	CALL_RESULT	8747	
	10	AE	7C	005AE		CLRQ	CALL_RESULT+8	8749	
59		56	D1	005B1		CMPL	STACK_PTR, UNDERFLOW_POINT	8751	
		15	1B	005B4		BLEQU	98\$		
	00000000'	EF	9F	005B6		PUSHAB	P.AFN		
		01	DD	005BC		PUSHL	#1		
	00028362	8F	DD	005BE		PUSHL	#164706		
00000000G	00	03	FB	005C4		CALLS	#3, LIB\$SIGNAL		
7E	59	56	C3	005CB	98\$:	SUBL3	STACK_PTR, UNDERFLOW_POINT, -(SP)	8753	
		56	DD	005CF		PUSHL	STACK_PTR		
		01	DD	005D1		PUSHL	#1	8752	
		01	DD	005D3		PUSHL	#1		
		52	DD	005D5		PUSHL	TEMP_THUNK_ADDR		
	1C	AE	9F	005D7		PUSHAB	CALL_RESULT		
0000V	CF	06	FB	005DA		CALLS	#6, VALSPEC ROUT_CALL		
	56	10	C2	005DF		SUBL2	#16, STACK_PTR	8754	
	5A	56	D1	005E2		CMPL	STACK_PTR, OVERFLOW_POINT		
		15	1E	005E5		BGEQU	99\$		
	00000000'	EF	9F	005E7		PUSHAB	P.AFO		
		01	DD	005ED		PUSHL	#1		
	00028362	8F	DD	005EF		PUSHL	#164706		
00000000G	00	03	FB	005F5		CALLS	#3, LIB\$SIGNAL		
66	08	AE	10	28	005FC	99\$:	MOVC3	#16, CALL_RESULT, (STACK_PTR)	8755
	0C	BC	00	D0	00601	100\$:	MOVL	DBG\$REG_VALUES+52, @FRAMEPTR	8756
			55	11	00609		BRB	106\$	8757
		56	04	C2	0060B	101\$:	SUBL2	#4, STACK_PTR	8777
		5A	56	D1	0060E		CMPL	STACK_PTR, OVERFLOW_POINT	
			15	1E	00611		BGEQU	102\$	
	00000000'	EF	9F	00613		PUSHAB	P.AFP		
		01	DD	00619		PUSHL	#1		
	00028362	8F	DD	0061B		PUSHL	#164706		
00000000G	00	03	FB	00621		CALLS	#3, LIB\$SIGNAL		
	00000000G	00	DD	00628	102\$:	PUSHL	DBG\$GL_CURRENT_PRIMARY	8778	
	C76A	CF	01	FB	0062E		CALLS	#1, DBG\$GET_OUTER_REC_ADDRESS	
			28	11	00633		BRB	105\$	
		56	04	C2	00635	103\$:	SUBL2	#4, STACK_PTR	8791
		5A	56	D1	00638		CMPL	STACK_PTR, OVERFLOW_POINT	
			15	1E	0063B		BGEQU	104\$	
	00000000'	EF	9F	0063D		PUSHAB	P.AFO		
		01	DD	00643		PUSHL	#1		
	00028362	8F	DD	00645		PUSHL	#164706		
00000000G	00	03	FB	0064B		CALLS	#3, LIB\$SIGNAL		
	00000000G	00	DD	00652	104\$:	PUSHL	DBG\$GL_CURRENT_PRIMARY	8792	
	C733	CF	01	FB	00658		CALLS	#1, DBG\$GET_INNER_REC_ADDRESS	
		66	50	D0	0065D	105\$:	MOVL	R0, (STACK_PTR)	
			57	D6	00660	106\$:	INCL	INSTRUC	8793
			F9BE	31	00662		BRW	1\$	8221
	08	BC	56	D0	00665	107\$:	MOVL	STACK_PTR, @RESULT_PTR	8809
			04	00669		RET		8812	

; Routine Size: 1642 bytes, Routine Base: DBG\$CODE + 33D1

```

: 8728      8813 1 ROUTINE VALSPEC_ERROR_HANDLER(SIGARG, MECHARG, ENBLARG) =
: 8729      8814 1
: 8730      8815 1 FUNCTION
: 8731      8816 1     This routine is the error handler for the DBG$STA_VALSPEC routine. It
: 8732      8817 1     handles Access Violations which occur during the evaluation of DSI Value
: 8733      8818 1     Specs. Since such access violations are not normally caused by errors
: 8734      8819 1     in Debug but rather by errors in the user program (e.g., by clobbered
: 8735      8820 1     registers), we give a special message for this kind of access violation.
: 8736      8821 1     The message says that the error occurred in the address computation for
: 8737      8822 1     some symbol and gives the symbol name. The symbol name comes from the
: 8738      8823 1     SYMID last passed to DBG$STA_SETCONTEXT.
: 8739      8824 1
: 8740      8825 1 INPUTS
: 8741      8826 1     SIGARG - The signal argument vector.
: 8742      8827 1
: 8743      8828 1     MECHARG - The mechanism argument vector.
: 8744      8829 1
: 8745      8830 1     ENBLARG - The enable argument vector (not used here).
: 8746      8831 1
: 8747      8832 1 OUTPUTS
: 8748      8833 1     For the SS$_ACCVIO error, the DBG$_ACCADDCOM error is signalled instead.
: 8749      8834 1     For all other errors, this routine just resignals.
: 8750      8835 1
: 8751      8836 1 BEGIN
: 8752      8837 2
: 8753      8838 2 MAP
: 8754      8839 2     SIGARG: REF VECTOR[.LONG];           ! Pointer to the signal argument vector
: 8755      8840 2
: 8756      8841 2 LOCAL
: 8757      8842 2     PATHDESCR,                       ! Pointer to pathname descriptor
: 8758      8843 2     PATHSTRING;                     ! Pointer to pathname string for symbol
: 8759      8844 2
: 8760      8845 2
: 8761      8846 2
: 8762      8847 2
: 8763      8848 2     ! If this is anything other than an access violation, just resignal it.
: 8764      8849 2
: 8765      8850 2 IF .SIGARG[1] NEQ SS$_ACCVIO THEN RETURN SS$_RESIGNAL;
: 8766      8851 2
: 8767      8852 2
: 8768      8853 2     ! It is an access violation. Determine the name of the last symbol passed
: 8769      8854 2     ! to DBG$STA_SETCONTEXT to set up the register context and use that in the
: 8770      8855 2     ! error message we substitute.
: 8771      8856 2
: 8772      8857 2 IF .DBG$REG_SYMID EQL 0
: 8773      8858 2 THEN
: 8774      8859 2     PATHSTRING = UPLIT BYTE(%ASCIC 'object')
: 8775      8860 2
: 8776      8861 2 ELSE
: 8777      8862 2     BEGIN
: 8778      8863 2         DBG$STA_SYMPATHNAME(.DBG$REG_SYMID, PATHDESCR);
: 8779      8864 2         DBG$NPATHDESC_TO_CS(.PATHDESCR, PATHSTRING);
: 8780      8865 2     END;
: 8781      8866 2
: 8782      8867 2
: 8783      8868 2     ! Signal the substitute error. We never get control back from the signal.
: 8784      8869 2
```

```
: 8785      8870 2  SIGNAL(DBG$_ACCADD COM, 1, .PATHSTRING);
: 8786      8871 2  RETURN 0;
: 8787      8872 2
: 8788      8873 1  END;
```

```
.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
74 63 65 6A 62 6F 06 00937 P.AFS: .ASCII <6>\object\
```

```
.PSECT DBG$CODE,NOWRT, SHR, PIC,0
0000 00000 VALSPEC_ERROR_HANDLER:
5E          08 C2 00002 .WORD Save nothing      : 8813
50          04 AC D0 00005 .SUBL2 #8, SP      : 8850
0C          04 A0 D1 00009 .MOVL SIGARG, R0
50          06 13 0000D .Cmpl 4(R0), #12
50          0918 8F 3C 0000F .BEQL 1$
50 00000000' 04 00014 .MOVZWL #2328, R0
50 00000000' EF D0 00015 1$: .RET
04 AE 00000000' 0A 12 0001C .MOVL DBG$REG_SYMID, R0      : 8857
16 11 00026 .BNEQ 2$
E223 CF 4001 8F BB 00028 2$: .MOVAB P.AFS, PATHSTRING      : 8859
04 AE 9F 00031 .BRB 3$
00000000G 00 02 FB 0002C .PUSHR #^M<R0,SP>      : 8863
04 AE DD 00034 .CALLS #2, DBG$STA_SYMPATHNAME      : 8864
00000000G 00 02 FB 00037 .PUSHAB PATHSTRING
04 AE DD 0003E 3$: .PUSHL PATHDESCR
01 DD 00041 .CALLS #2, DBG$NPATHDESC_TO_CS
00000000G 00 8F DD 00043 .PUSHL PATHSTRING      : 8870
03 FB 00049 .PUSHL #1
04 00052 .PUSHL #167064
03 FB 00049 .CALLS #3, LIB$SIGNAL
04 00050 .CLRL R0
04 00052 .RET      : 8871
: 8873
```

; Routine Size: 83 bytes, Routine Base: DBG\$CODE + 3A3B


```

: 8790      8874 1 ROUTINE VALSPEC_SCOPE_ERROR: NOVALUE =
: 8791      8875 1
: 8792      8876 1 FUNCTION
: 8793      8877 1 This routine is called during DST Value Spec evaluation if a register
: 8794      8878 1 is referenced which is not available in the current context as set by
: 8795      8879 1 routine DBG$STA_SETCONTEXT. Use of such a register usually means that
: 8796      8880 1 a variable is being referenced whose scope is not currently active, i.e.
: 8797      8881 1 there is no CALL frame on the VAX stack for the routine in which the
: 8798      8882 1 symbol is declared. This routine just sets up and signals the "Symbol
: 8799      8883 1 not in active scope" error message.
: 8800      8884 1
: 8801      8885 1 INPUTS
: 8802      8886 1 DBG$REG_SYMID is an implicit input. It gives the SYMID of the symbol
: 8803      8887 1 last used to establish context. There are no input parameters.
: 8804      8888 1
: 8805      8889 1 OUTPUTS
: 8806      8890 1 NONE
: 8807      8891 1
: 8808      8892 1
: 8809      8893 2 BEGIN
: 8810      8894 2
: 8811      8895 2 LOCAL
: 8812      8896 2 PATHNAME, ! Pointer to symbol's pathname descriptor
: 8813      8897 2 PATHSTRING; ! Pointer to symbol's pathname string
: 8814      8898 2
: 8815      8899 2
: 8816      8900 2
: 8817      8901 2 ! Use the SYMID passed to DBG$STA_SETCONTEXT last to format the symbol name
: 8818      8902 2 ! for the error message. If no such name exists, use the null string.
: 8819      8903 2
: 8820      8904 2 IF .DBG$REG_SYMID EQL 0
: 8821      8905 2 THEN
: 8822      8906 2 PATHSTRING = UPLIT(0)
: 8823      8907 2
: 8824      8908 2 ELSE
: 8825      8909 2 BEGIN
: 8826      8910 2 DBG$STA_SYMPATHNAME(.DBG$REG_SYMID, PATHNAME);
: 8827      8911 2 DBG$NPATHDESC_TO_CS(.PATHNAME, PATHSTRING);
: 8828      8912 2 END;
: 8829      8913 2
: 8830      8914 2
: 8831      8915 2 ! Signal the error--we do not return from the signal.
: 8832      8916 2 !
: 8833      8917 2 SIGNAL(DBG$_SYMNOTACT, 1, .PATHSTRING);
: 8834      8918 2
: 8835      8919 1 END;
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

00000000 0093E .BLKB 2
00940 P.AFT: .LONG 0

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

0000 00000 VALSPEC_SCOPE_ERROR:					
	SE		08 C2 00002	WORD Save nothing	: 8874
	50	00000000	EF D0 00005	SUBL2 #8, SP	: 8904
			0A 12 0000C	MOVL DBG\$REG_SYMID, R0	: 8906
04	AE	00000000	EF 9E 0000E	BNEQ 1\$: 8910
		4001	16 11 00016	MOVAB P.AFT, PATHSTRING	: 8911
E1E0	CF		8F BB 00018 1\$:	BRB 2\$: 8917
		04	02 FB 0001C	PUSHR #^M<R0, SP>	: 8919
		04	AE 9F 00021	CALLS #2, DBG\$STA_SYMPATHNAME	
00000000G	00		AE DD 00024	PUSHAB PATHSTRING	
		04	02 FB 00027	PUSHL PATHNAME	
			AE DD 0002E 2\$:	CALLS #2, DBG\$NPATMDISC_TO_CS	
		00028C88	01 DD 00031	PUSHL PATHSTRING	
00000000G	00		8F DD 00033	PUSHL #1	
			03 FB 00039	PUSHL #167048	
			04 00040	CALLS #3, LIB\$SIGNAL	
				RET	: 8919

; Routine Size: 65 bytes, Routine Base: DBG\$CODE + 3A8E

```
8837 8920 1 ROUTINE VALSPEC_ROUT_CALL( VALBUFFER,  
8838 8921 1 ROUT_ADDR,  
8839 8922 1 OCTAWORD_FLAG,  
8840 8923 1 FP_FLAG,  
8841 8924 1 STACK_TOP,  
8842 8925 1 STACK_LENGTH) : NOVALUE =  
8843 8926 1  
8844 8927 1 FUNCTION  
8845 8928 1 This routine is called to handle calls on compiler-supplied routines  
8846 8929 1 in the user's address-space during Value Spec evaluation. Calls to  
8847 8930 1 compiler-supplied Value Spec routines can be specified in Materializa-  
8848 8931 1 tion Specs in Value Specs, both directly and via the DST Stack Machine.  
8849 8932 1 The compiler-supplied routine is called as follows:  
8850 8933 1  
8851 8934 1 - The desired symbol's Frame Pointer value is passed to  
8852 8935 1 the routine in register R1.  
8853 8936 1  
8854 8937 1 - If OCTAWORD_FLAG is FALSE, a pointer to the vector of  
8855 8938 1 register values for the symbol's frame (as represented  
8856 8939 1 by DBGSREG_VALUES) is passed as a parameter in the argu-  
8857 8940 1 ment vector, and the routine returns the symbol's value  
8858 8941 1 in register R0.  
8859 8942 1  
8860 8943 1 - If OCTAWORD_FLAG is TRUE, a pointer to a 4-longword result  
8861 8944 1 buffer and a pointer to the vector of register values in  
8862 8945 1 the symbol's frame are passed as parameters in the argument  
8863 8946 1 vector. The routine's result is returned directly to the  
8864 8947 1 result buffer in this case, and not through register R0.  
8865 8948 1  
8866 8949 1 - When STACK_TOP and STACK_LENGTH are passed they are passed  
8867 8950 1 as the 2nd and 3rd parameters if the OCTAWORD_FLAG is false  
8868 8951 1 and the 3rd and 4th parameters if the OCTAWORD_FLAG is true.  
8869 8952 1  
8870 8953 1 If the Frame Pointer (FP) is not available in the current context (as  
8871 8954 1 set up by DBG$STA SETCONTEXT), the "symbol not in active scope" error  
8872 8955 1 is signalled. Otherwise the compiler-supplied routine is called as  
8873 8956 1 described above and its value returned. The routine that called  
8874 8957 1 VALSPEC_ROUT_CALL can then use the value as it sees fit.  
8875 8958 1  
8876 8959 1 INPUTS  
8877 8960 1 VALBUFFER - The address of a 1-longword or 4-longword buffer which is  
8878 8961 1 to receive the value returned by the called routine. The  
8879 8962 1 size of the buffer depends on the value of OCTAWORD_FLAG.  
8880 8963 1 The buffer should be zeroed out by the caller.  
8881 8964 1  
8882 8965 1 ROUT_ADDR - The address of the routine to be called to get the value.  
8883 8966 1  
8884 8967 1 OCTAWORD_FLAG - A flag value set to TRUE if the called routine is  
8885 8968 1 expected to return a 4-longword value to VALBUFFER. If  
8886 8969 1 this flag is FALSE, a single longword is expected to be  
8887 8970 1 returned to VALBUFFER. If OCTAWORD_FLAG is TRUE, the called  
8888 8971 1 routine is expected to return its value to the address given  
8889 8972 1 by the first parameter; otherwise, the value is returned in  
8890 8973 1 register R0.  
8891 8974 1  
8892 8975 1 FP_FLAG - If TRUE, indicates that FP is to be passed in to thunk.  
8893 8976 1
```

```
8894 8977 1  STACK_TOP - Optional parameter. Pointer to the top of the stack
8895 8978 1  in the stack machine. Passed by value.
8896 8979 1
8897 8980 1  STACK_LENGTH - Optional parameter. The number of bytes on the stack
8898 8981 1  in the stack machine. Passed by value.
8899 8982 1
8900 8983 1  OUTPUTS
8901 8984 1  VALBUFFER - The value returned by the called compiler-supplied
8902 8985 1  routine is returned to the buffer pointed to by
8903 8986 1  VALBUFFER.
8904 8987 1
8905 8988 1  No routine value is returned.
8906 8989 1
8907 8990 1
8908 8991 2  BEGIN
8909 8992 2
8910 8993 2  BUILTIN
8911 8994 2  ACTUALCOUNT,
8912 8995 2  ACTUALPARAMETER;
8913 8996 2
8914 8997 2  MAP
8915 8998 2  VALBUFFER: REF VECTOR[,LONG]; ! Pointer to buffer to receive value
8916 8999 2
8917 9000 2  ENABLE
8918 9001 2  VALSPEC_ROUT_CALL_HANDLER; ! Set up a handler for this routine
8919 9002 2
8920 9003 2
8921 9004 2  ! Define the linkage by which we call the compiler-supplied routine.
8922 9005 2
8923 9006 2  LINKAGE
8924 9007 2  ROUT_CALL_LINKAGE = CALL(REGISTER = 1, STANDARD);
8925 9008 2
8926 9009 2  BIND ROUTINE
8927 9010 2  ROUTINE_TO_CALL = (.ROUT_ADDR): ROUT_CALL_LINKAGE;
8928 9011 2
8929 9012 2  !++
8930 9013 2  ! 4 parameters not allowed
8931 9014 2  !--
8932 9015 2  IF ACTUALCOUNT() EQL 5
8933 9016 2  THEN
8934 9017 2  $DBG_ERROR('RSTACCESS\VALSPEC_ROUT_CALL');
8935 9018 2
8936 9019 2
8937 9020 2  ! Make sure there is a current register set to take FP from.
8938 9021 2
8939 9022 2  IF .FP_FLAG
8940 9023 2  THEN
8941 9024 2  IF .DBG$REG_VECTOR[13] EQL 0 THEN VALSPEC_SCOPE_ERROR();
8942 9025 2
8943 9026 2
8944 9027 2  ! Call the compiler-provided routine to compute the desired value. If the
8945 9028 2  ! octaword flag is set, we pass the buffer to receive the value (up to four
8946 9029 2  ! longwords) as the first parameter. Otherwise we get the value from R0.
8947 9030 2  ! The frame pointer value is always passed in in register R1.
8948 9031 2
8949 9032 2  IF ACTUALCOUNT() EQL 4
8950 9033 2  THEN
```

```
: 8951      9034      2      IF .OCTAWORD_FLAG
: 8952      9035      2      THEN
: 8953      9036      2      ROUTINE_TO_CALL(.DBG$REG_VALUES[13], .VALBUFFER, DBG$REG_VALUES[0])
: 8954      9037      2      ELSE
: 8955      9038      2      VALBUFFER[0] = ROUTINE_TO_CALL(.DBG$REG_VALUES[13], DBG$REG_VALUES[0])
: 8956      9039      2      ELSE
: 8957      9040      2      IF .OCTAWORD_FLAG
: 8958      9041      2      THEN
: 8959      9042      2      ROUTINE_TO_CALL(.DBG$REG_VALUES[13], .VALBUFFER, DBG$REG_VALUES[0], .STACK_TOP, .STACK_LENGTH)
: 8960      9043      2      ELSE
: 8961      9044      2      VALBUFFER[0] = ROUTINE_TO_CALL(.DBG$REG_VALUES[13], DBG$REG_VALUES[0], .STACK_TOP, .STACK_LENGTH)
: 8962      9045      2
: 8963      9046      2      RETURN;
: 8964      9047      2
: 8965      9048      1      END;
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

```
53 4C 41 56 5C 53 53 45 43 43 41 54 53 52 1B 00944 P.AFU: .ASCII <27>\RSTACCESS\<92>\VALSPEC_ROUT_CALL\
4C 4C 41 43 5F 54 55 4F 52 5F 43 45 50 00953
```

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

```
000C 00000 VALSPEC_ROUT_CALL:
: 8920      53 00000000G 00 9E 00002      .WORD      Save R2,R3
: 8991      6D      0077  CF DE 00009      MOVAB      DBG$REG_VALUES, R3
: 9015      05      6C 91 0000E      MOVAL      7$, (FPT)
: 9017      00000000' EF 9F 00013      CMPB      (AP), #5
: 9022      00028362 8F DD 0001B      BNEQ      1$
: 9024      00000000G 00 03 FB 00021      PUSHAB     P.AFU
: 9036      0C      10 AC E9 00028 1$:      PUSHL     #1
: 9032      00000000G 00 04 12 00032      PUSHL     #164706
: 9034      87 AF      00 00 FB 00034      CALLS     #3, LIB$SIGNAL
: 9036      52      34 A3 D0 00038 2$:      BLBC     FP_FLAG, 2$
: 9038      04      6C 91 0003C      TSTL     DBG$REG_VECTOR+52
: 9042      0D      0C AC E9 00041      BNEQ     2$
: 9044      04      53 DD 00045      CALLS     #0, VALSPEC_SCOPE_ERROR
: 9046      51      04 AC DD 00047      MOVL     DBG$REG_VALUES+52, R2
: 9038      08 BC      02 04 00051      CMPB     (AP), #4
: 9042      51      53 DD 00052 3$:      BNEQ     4$
: 9044      08 BC      01 FB 00057      BLBC     OCTAWORD_FLAG, 3$
: 9046      11      0C AC E9 0005D 4$:      PUSHL     R3
: 9048      7E      14 AC 7D 00061      PUSHL     R3
: 9050      53 DD 00065      MOVL     VALBUFFER
: 9052      51      52 D0 00054      MOVL     R2, R1
: 9054      08 BC      01 FB 00057      CALLS     #1, @ROUT_ADDR
: 9056      11      0C AC E9 0005D      BRB      6$
: 9058      7E      14 AC 7D 00061      BLBC     OCTAWORD_FLAG, 5$
: 9060      53 DD 00065      MOVQ     STACK_TOP, -(SP)
: 9062      53 DD 00065      PUSHL     R3
```

		04	AC	DD	00067		PUSHL	VALBUFFER		
	51		52	DD	0006A		MOVL	R2, R1		
08	BC		04	FB	0006D		CALLS	#4, @ROUT_ADDR		
					04	00071	RET			
	7E	14	AC	7D	00072	5\$:	MOVQ	STACK_TOP, -(SP)	9044	
			53	DD	00076		PUSHL	R3		
	51		52	DD	00078		MOVL	R2, R1		
08	BC		03	FB	0007B		CALLS	#3, @ROUT_ADDR		
04	BC		50	DD	0007F	6\$:	MOVL	R0, @VALBUFFER		
					04	00083	RET		9048	
					0000	00084	7\$:	.WORD	Save nothing	8991
			7E	DD	00086		CLRL	-(SP)		
			5E	DD	00088		PUSHL	SP		
	7E	04	AC	7D	0008A		MOVQ	4(AP), -(SP)		
0000V	CF		03	FB	0008E		CALLS	#3, VALSPEC_ROUT_CALL_HANDLER		
					04	00093	RET			

; Routine Size: 148 bytes, Routine Base: DBG\$CODE + 3ACF

```

: 8967 9049 1 ROUTINE VALSPEC_ROUT_CALL_HANDLER(SIGARG, MECHARG, ENBLARG) =
: 8968 9050 1
: 8969 9051 1 FUNCTION
: 8970 9052 1 This routine is the error handler for the VALSPEC_ROUT_CALL routine. It
: 8971 9053 1 handles abnormal conditions which occur during the evaluation of
: 8972 9054 1 PLI Base Variable, ie. Pointer to the base variable has not been set
: 8973 9055 1 up by the ALLOCATE in PLI before the program execution. However we
: 8974 9056 1 do allow symbol is not active signal to go through this routine.
: 8975 9057 1
: 8976 9058 1 INPUTS
: 8977 9059 1 SIGARG - The signal argument vector.
: 8978 9060 1
: 8979 9061 1 MECHARG - The mechanism argument vector.
: 8980 9062 1
: 8981 9063 1 ENBLARG - The enable argument vector (not used here).
: 8982 9064 1
: 8983 9065 1 OUTPUTS
: 8984 9066 1 For the DBGS_SYMNOTACT error, this routine just resignal.
: 8985 9067 1 For all other errors, this routine signals DBGS_BASVARNOTSET.
: 8986 9068 1
: 8987 9069 1
: 8988 9070 2 BEGIN
: 8989 9071 2
: 8990 9072 2 MAP
: 8991 9073 2 SIGARG: REF VECTOR[.LONG]; ! Pointer to the signal argument vector
: 8992 9074 2
: 8993 9075 2
: 8994 9076 2 IF .SIGARG[1] EQL DBGS_SYMNOTACT OR
: 8995 9077 2 .SIGARG[1] EQL $$$_ONWIND
: 8996 9078 2 THEN
: 8997 9079 2 RETURN $$$_RESIGNAL;
: 8998 9080 2
: 8999 9081 2 SIGNAL(DBGS_BASVARNOTSET);
: 9000 9082 2 RETURN 0;
: 9001 9083 2
: 9002 9084 1 END;
```

```

                                0000 0000 VALSPEC_ROUT_CALL_HANDLER:
                                .WORD Save nothing
00028C88 50 04 AC D0 00002 MOVL SIGARG, R0
                                9049
                                8F 04 A0 D1 00006 CMPL 4(R0), #167048
                                9076
00000920 8F 04 A0 D1 0000E BEQL 1$
                                9077
                                06 12 00018 CMPL 4(R0), #2336
                                9079
                                50 0918 8F 3C 0001A 1$: MOVZWL #2328, R0
                                04 0001F RET
                                00028108 8F DD 00020 2$: PUSHL #164104
                                00 01 FB 00026 CALLS #1, LIB$SIGNAL
                                9081
                                50 D4 0002D CLRL R0
                                04 0002F RET
                                9082
                                9084
```

; Routine Size: 48 bytes, Routine Base: DBGS\$CODE + 3863

: 9003 9085 1
: 9004 9086 1
: 9005 9087 0 END ELUDOM

.EXTRN LIB\$SIGNAL, SYSSUNWIND

PSECT SUMMARY

Name	Bytes	Attributes
DBG\$OWN	92	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$PLIT	2400	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)
DBG\$CODE	15251	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	20	0	1000	00:01.9
_\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	3	9	7	00:00.1
_\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	224	14	97	00:02.0
_\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1	418	233	55	31	00:00.3
_\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	14	3	22	00:00.3
_\$255\$DUA28:[DEBUG.OBJ]DBGGEN.L32;1	150	1	0	12	00:00.3

COMMAND QUALIFIERS

: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:RSTACCESS/OBJ=OBJ\$:RSTACCESS MSRC\$:RSTACCESS/UPDATE=(ENH\$:RSTACCESS)

: Size: 15251 code + 2492 data bytes
: Run Time: 04:32.7
: Elapsed Time: 05:19.3
: Lines/CPU Min: 1999
: Lexemes/CPU-Min: 15128
: Memory Used: 876 pages
: Compilation Complete

0098

AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

0099 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

